DOCUMENT RESUME

ED 385 968 EC 304 162

AUTHOR Woodward, John; Gersten, Russell
TITLE The TORUS Project: An Innovative-Assessment Technology Program. Final Report.
INSTITUTION Eugene Research Inst., OR.
PUB DATE 30 Mar 92
CONTRACT H180B90020-90
NOTE 93p.
PUB TYPE Reports - Descriptive (141) -- Reports - Evaluative/Feasibility (142)

EDRS PRICE MF01/PC04 Plus Postage.
DESCRIPTORS Addition; *Computer Managed Instruction; *Computer Software; Construct Validity; *Elementary School Mathematics; *Error Patterns; Intermediate Grades; Junior High Schools; *Learning Disabilities; Material Development; *Mathematics Instruction; Middle Schools; Program Evaluation; Subtraction

ABSTRACT

This report describes TORUS, a computer-based program for elementary school students with learning disabilities which analyzes student work samples in addition and subtraction and provides individual profiles of student misconceptions and weaknesses. Initially, work samples from 236 middle school students with learning disabilities were analyzed by a human expert to assess the relative difficulty of addition and subtraction in terms of number and types of misconceptions and to provide an empirical basis for encoding misconception algorithms into the TORUS program. Pilot test data, construct validity research, and criterion validity studies are described, as are the use of commercial programs as building blocks for sophisticated software and differences between TORUS and the BUGGY program. The 10 most common substraction misconceptions are identified and matched to the sequence of the "Corrective Mathematics Program," a remedial curricula for special education students. Included in the report are a sample TORUS student profile and the following two papers: "A Comparison of TORUS and Human Expert Diagnoses: A Construct Validity Study" (Michael Landes) and "Software Development in Special Education" (John Woodward). (Contains 109 references.) (SW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED 385 968

The TORUS Project: An Innovative-Assessment Technology Program

Final Report

H180B90020-90

Project Directors

John Woodward

Russell Gersten

Eugene Research Institute
1400 High Street
Eugene, Oregon 97401

March 30, 1992

BEST COPY AVAILABLE

EC 304162

Table of Contents

		page
1.0	Introduction to the Final Report	1
2.0	Pilot Data and Preliminary Analyses	3
3.0	A Comparison of TORUS and Human Expert Diagnoses: A Construct Validity Study	7
4.0	Software Development in Special Education (*in preparation*)	32
5.0	Detailed Comparison of BUGGY and TORUS	64
6.0	Remedying Misconceptions: Coordinating TORUS Disagnoses with SRA's Corrective Mathematics Program	78
7.0	Current Uses of the TORUS program	84
8.0	Dissemination	85

1.0 Introduction to the Final Report

The Expert Analysis of Mathematics Proficiency (EXAMPLE) is an innovative use of technology to assess student misconceptions in addition and subtraction. Shortly after the inception of the project, the program was renamed TORUS, which is the term that is used throughout this report. The TORUS program focuses on the addition and subtraction operations because of the "well-structured" nature of these operations (i.e., they are amenable to fine grained analyses through artificial intelligence programs). In this respect, these operations are a vehicle for testing the limits of sophisticated computer analyses of student error patterns.

In the initial phases of the project, work samples from 236 middle school students with learning disabilities were analyzed by a human expert. The purpose of this was twofold: a) to assess the relative difficulty of addition and subtraction in terms of number and type of misconceptions and b) to provide an empirical basis for encoding misconception or "bug" algorithms into the TORUS program. Pilot test data, preliminary construct validity research, and criterion validity studies are described in Section 2.0 of this report.

Results from the human-expert analyses, pilot tests of TORUS, and a further review of the literature on misconceptions enabled the project directors and staff to formatively evaluate TORUS as an innovative technology assessment program. These individuals concluded at this point in the project that: 1) subtraction was a far more complicated operation than addition, one which produced many more student misconceptions 2) subtraction misconceptions were insensitive to the kind of remedial instruction commonly found in special education classrooms 3) construct validity studies were crucial to TORUS as a viable diagnostic tool and 4) the commercial programs used to create TORUS

(i.e., FoxBase™ and Nexpert Object™) were highly robust tools for developing high quality software under highly constrained development timelines. Further construct and social validity studies of TORUS are described in Section 3.0. The use of commercial programs as building blocks for sophisticated software is described in Section 4.0.

The final TORUS program is a comprehensive tool for diagnosing subtraction misconceptions and prototypes a similar kind of diagnosis for addition. The algorithms used in TORUS are empirically derived, and thus, of much greater potential use to diagnosticians and teachers than previous programs such as BUGGY (Brown & Burton, 1978). Differences between BUGGY and TORUS are described in Section 5.0.

TORUS diagnoses have also been linked to a highly popular remedial mathematics curricula used in special education classrooms -- SRA's Corrective Mathematics Program. One of the senior authors of this curricula carefully analyzed it's scope and sequence, correlating it to the TORUS diagnoses. The final remediation guide, which appears in Section 6.0, streamlines a student's path through the Corrective Mathematics Program once a diagnosis has been rendered. The final two sections of this report describe the current uses of TORUS (Section 7.0) as well as dissemination activities (Section 8.0) which have been conducted over the last two years.

An actual copy of the TORUS program has not been enclosed with this report because its operation depends on two commercial software programs (i.e., FoxBase™ and Nexpert Object™). It would be a violation of copyright protection laws to distribute copies of these programs in this fashion. However, a beta version of TORUS was demonstrated to Drs. Martin Kaufmann, Tom Hanley, Martha Couthino, and Kenneth Denny during a site visit to the Eugene Research Institute in May 1991.

2.0 Pilot Data and Preliminary Analyses

The TORUS program (Woodward, Freeman, Blake, & Howard, 1990) was explicitly designed to build upon the work conducted in the BUGGY project (J. S. Brown & Burton, 1978; J. S. Brown & VanLehn, 1982; Burton, 1981; VanLehn, 1982, 1988, 1990). Like BUGGY, TORUS captures the subtleties of student misconceptions in a systematic manner through "after the fact" analyses of responses. In the spirit of IDEBUGGY (Burton, 1981), TORUS interactively generates problems to test hypotheses about sources of students' errors. The hypotheses are derived from a thorough analysis of a standard set of 30 addition or 40 subtraction problems and confirmed with a follow-up set of problems. For each student, the system renders an individual diagnosis that explains the majority of that student's errors. Educational prescriptions tailored to the pragmatic needs of diagnosticians and special educators can accompany each diagnosis.

Hardware and Software Features

TORUS has been designed for the Macintosh II line of computers. The decision to use this computer was made on the basis of Apple Computer's preeminence in the field of education as well as the likelihood that these computers will continue to replace the aging supply of Apple II type computers in public schools. The Macintosh II computer has also enabled us to use Neuron Data's Nexpert Object expert system shell and FoxBase's relational database management system as core technologies. Each program is extremely powerful, widely used across a range of microcomputer and mainframe platforms, and recognized as a leader in its field internationally. This software development strategy has allowed us to build a very sophisticated program in a relatively short period of time; one that is reliable

and free of the erratic behavior of programs that are written from "scratch" in a programming language such as Pascal or C+.

At an early stage in the development of the project it was decided that many issues of ecological validity would be obviated by having the students work the problems with pencil and paper rather than directly on the computer. By using teachers or other trained personnel to enter the student performance data, one computer can provide information about the performance of any number of students.

Analyzing Misconceptions in Two Diagnostic Sessions

The analyses of addition or subtraction occur across two sequential sessions in TORUS. The first part of the test (The TORUS Achievement Test) consists of 40 problems designed to reliably assess student computational performance. The test covers a reasonable range of problems for the intermediate and middle school, learning disabled student and has four alternate forms. Coefficient alpha reliability for the test is .97. There are four alternate forms to this test (alternate form reliability = .93).

The TORUS analysis yields hypotheses about overall competence, the areas of difficulty and proficiency, and whether or not there are "bugs" (i.e., misconceptions that result in specific student behavior). The second test (The Profile of Misconceptions) consists of problems selected by the expert system to confirm/disconfirm the hypotheses established by the first session.

For example, if the first session analysis indicates a high error rate on subtraction problems that contain double zero borrows (e.g., 4007-831), the second session test would contain (a) problems of the same type as those missed-- i.e., double zero borrows, and (b) problems that are minimally different-- e.g., single zero borrows (407-31), and double non-zero borrows (4657-982)-- in order to confirm or disconfirm the hypothesis. This kind of

hypothesis testing through new data is a common feature of expert systems technology (Bowerman & Glover, 1988; Hayes-Roth, Waterman, & Lenat, 1983; Rich, 1983; Wenger, 1987).

As a final step in the analysis, bug algorithms similar to those employed in BUGGY are applied to the incorrect problems. If a given bug algorithm yields the same answers to the problems as the student, the inference is made that the student has that bug. The number of bug algorithms has been limited to the 30 most frequently documented in the literature. These misconceptions account for over 80 percent of student errors that can be identified as buggy in our research as well as that of Brown and Burton (1978) and Young and O'Shea (1981). The number of bugs has been restricted because it is felt that low incidence bugs have little educational meaning to diagnosticians or special educators.

Construct Validity

The final diagnostic summary of performance specifies: (a) the percent correct on the total test; (b) the percent correct and incorrect on different problem types; (c) and any specific procedural errors found. Human expert agreement with the TORUS program on these three levels of diagnosis is 1.0 for percent correct, .95 for percent incorrect by problem type, and .90 for final error diagnosis. All three levels of diagnosis are helpful to the researcher, diagnostician, and the practitioner. This kind of information effectively profiles competence as teachers proceed through a standard addition or subtraction curriculum -- beginning with the least complex (e.g., non-borrow problems) and working toward more complex ones (e.g., subtraction problems involving zeros and multiple borrows).

Criterion Validity

Two studies were conducted to assess the criterion-related validity of TORUS to the Metropolitan Achievement Test (MAT). Seventy-five middle school students with learning disabilities took the TORUS Addition Test and the Computations Subtest. Correlations between raw scores and the MAT were .44 ($p < .001$). At the same time, 71 of these students took the TORUS Subtraction Test. Similar analyses were conducted with the raw scores, yielding a correlation of .51 ($p < .001$).

Another set of correlations were conducted in the Fall 1992 with 155 intermediate and middle school students with the MAT and the TORUS Subtraction Test. Again, there were signficant correlations between raw scores on each ($p < .01$), as well as general misconceptions ($p < .01$). This later correlation included students whose main misconception was the inability to borrow. However, on the more subtle misconceptions, there was no significant correlation.

3.0 A Comparison of TORUS and Human Expert Diagnoses: A Construct Validity Study

Michael Landes

Masters Project in Special Education
Mildly Handicapped Program
University of Oregon

Fall, 1991

Math assessment has been used to help teachers plan and make instructional decisions. When students do poorly on a test, changes in instruction are advisable. However, most teachers lack the knowledge to diagnose math error patterns (e.g., in subtraction) as well as a repertoire of alternate teaching strategies (Fuchs & Hamlett, 1988). When faced with low scores, teachers often re-teach in the same curriculum using the same text and teaching methods (Fuchs & Hamlett, 1988; Ysseldyke, O'Sullivan, Thurlow, & Christenson, 1989). McLeod and Armstrong (1982) found that teachers rarely provided strategy instruction, but relied on drill and extra practice using the basal text and supplemental worksheets. Woodward and Gersten (in press) reported similar findings.

It is likely that merely recycling students through curriculum in much the same way that it was taught the first time would do little to correct the difficulties and misconceptions causing the errors (Engelmann & Carnine, 1982). Furthermore, remedial instruction that simply provides extra practice does not remedy the underlying causes of math errors (Fuchs & Hamlett, 1988). Student misconceptions can persist when students do not have a conceptual understanding of the material.

Recent research (Howard & Woodward, 1990) suggests that some students who have received specialized math instruction for more than five years continue to show some kind of systematic pattern of errors in subtraction. Assessment is needed which permits analysis of a student's thinking process so that remediation can target the causes of the confusion. Errors that at first appear random actually may be the result of a student employing an incorrect algorithm (Brown & Burton, 1978). Assessment results should include more than the percentage of items worked correctly

and the type of problems missed. Clearly, there needs to be a change in how teachers assess and remedy math errors.

The remainder of this section outlines some of the drawbacks to current assessment methods, along with issues in cognitive assessment of math errors. Finally, it is argued that computers *can* aid classroom teachers in diagnosing student misconceptions in math if the information is sufficiently detailed.

Traditional Math Assessment

Although thorough diagnoses of student errors are recommended by some researchers (e.g., Silbert, Carnine, & Stein, 1981; Hammill, 1987), they are time consuming and rarely found in classroom practice. Furthermore, if teachers are asked to work an additional hour or two with no added compensation, few will do it (Duckworth & Fielding, 1985; Stevens & Driscoll, 1985).

On the other hand, when assessment systems are simplified (e.g., percent correct) other problems arise. Students who score at acceptable levels on math tests, for example, may actually have a weak understanding of the subject (Schoenfeld, 1985). Recent investigations (Howard & Woodward, 1990) of middle school students with learning disabilities in math showed that on average, students scored at the 70 percent level in subtraction. However, over half of the students also showed serious misconceptions about this operation. To find out if students understand the concepts and processes that form the core of math knowledge, assessment needs to provide information on how students are thinking when they work math problems.

Problems with Curriculum Based Measurement

Curriculum based measurement (CBM) has emerged as a testing measure closely aligned to classroom curriculum and therefore useful for

day-to-day instructional planning (Fuchs & Fuchs, 1986). CBM was developed because of dissatisfaction with standardized tests, which some researchers believed were not appropriate for everyday instructional planning (Algozzine & Ysseldyke, 1982; Glass, 1983; Salvia & Ysseldyke, 1985). Jenkins and Pany (1978) and Good and Salvia (1988) found a poor match between standardized tests and curricula used in classrooms. Also, standardized tests were little help to teachers in planning instructional strategies.

CBM assesses student growth across time using measures that cover material representative of the curriculum used in the classroom. Assessments are made frequently (e.g., every two weeks), giving teachers ongoing evaluation of instructional strategy and program success. By following actual student progress on a graph relative to an aim line (i.e., the expected rate of progress), the teacher has a guide for making instructional decisions. Frequent collection of student progress data should aid teachers in evaluating their interventions (Idol, 1986; Fuchs & Fuchs, 1986). The link between CBM and classroom curriculum makes CBM a useful tool for teachers decision making (Fuchs, Fuchs, Hamlett, & Stecker, 1991).

Curriculum based measurement does have its drawbacks. CBM usually include the range of problems in a year's curriculum. As the year progresses and students are taught additional problem types, their scores should rise accordingly. However, a student may be making general progress within a curriculum but have a specific weakness that is not fully documented by the test results. Furthermore, even though CBM show what problems are missed, the cause of the errors are still unknown to the teacher.

Another problem with CBM is that student growth is assumed to progress at an even rate throughout the year. It is unlikely that every student

will experience steady growth. Assessment of student errors needs to go beyond what is provided by CBM.

Informed Instruction

According to Brown and Campione (1986), informed instruction includes teacher understanding of students' cognitive processes, which leads to more effective remediation. Also, informed instruction incorporates effective teacher practices including explicit strategies, immediate feedback, and interactive teaching (Gersten, Woodward, & Darch, 1986; Woodward, 1991). Teachers who employ varied teaching strategies, have thorough knowledge of the content area, and understand how students are thinking about the material, are most likely to establish an effective instructional setting in the classroom.

Expert Assessment

Leinhardt and Smith (1985) found that expert teachers (i.e. teachers with superior levels of student achievement) had detailed diagnosis and "understanding of classes of student errors" and instructional plans to target those errors. Without more detailed information about how students are thinking, teachers can not match instruction to individual needs, and many students will continue to process math using incorrect strategies. Expert assessment is a necessary component of informed instruction. This suggests that math assessment needs to supply more diagnostic information than is provided by either standardized tests or current forms of CBM.

Computer Based Cognitive Assessment

BUGGY. BUGGY (Brown & Burton, 1978) was one of the first artificial intelligence programs to assess common mathematical knowledge. The system was designed to mimic the human cognitive processes used in solving subtraction problems. BUGGY's purpose was to provide more information

than percent of problems correct and what problem types were missed. Using its data base of possible subtraction errors, BUGGY provided a cognitive account of what caused the errors. Brown and Burton (1978) found that the BUGGY computer system enabled student teachers to discern students' error patterns and misconceptions from random errors.

The BUGGY project has been a major influence in the study of cognitive assessment and the use of technology in education over the past ten years (Carpenter, Moser, & Romberg, 1982; Cawley, 1985; Ferrara, 1987; Ginsburg, 1983; Pellegrino & Goldman, 1987; Resnick & Ford, 1981; Woodward & Carnine, 1988). The BUGGY research showed that students interpret what they are taught. Students created their own algorithms to work problems in subtraction even after they had been taught correct algorithms by a teacher. This research shed new light on the cognitive processes students use in working math problems.

Limitations of BUGGY. As a prototype computer assessment system BUGGY explored technological capability to simulate human thinking patterns with minimal regard for the instructional framework of classrooms. The subtraction problems presented to students were not tested for item reliability or validity. Some problems were too difficult and resulted in obscure error diagnoses and computer generated misconceptions.

The use of computers for student assessment should match the teacher's need for information. Too much data can leave teachers wondering what information is relevant. Brown and Burton (1978) found that the majority of student math errors were embedded in a few misrules. Computer assessment can be useful to teachers without providing detailed analyses of obscure error types. Computer assistance in assessment should blend with pragmatic classroom needs (Woodward & Carnine, 1988).

Computers can process information rapidly and deliver error analyses to teachers that are otherwise impractical to collect. Computers are a tool that can help teachers pinpoint misconceptions in math accurately and efficiently.

TORUS. TORUS is a computer based program for elementary school math assessment for students with learning disabilities (Woodward, Freeman, Blake, & Howard, 1990). TORUS was designed to analyze student work samples in addition and subtraction and provide individual profiles of student misconceptions and weaknesses.

The developers of TORUS built upon previous research of computer based assessment systems (e.g., Brown & Burton, 1978). BUGGY has been referred to in the research literature as a prototype of a system that will bring together assessment and remedial instruction (Brown & Campione, 1986).

The link between a teacher's ability to make informed instructional decisions and the efficient gathering of precise information on a student's math thinking was at the heart of the TORUS design. The goal was to mesh technology with the realistic needs of classroom teachers. This philosophy corresponds with the concept of informed instruction (Brown & Campione, 1986) that teacher understanding of the causes for student errors is necessary to provide appropriate remediation instruction.

Another improvement in the design of TORUS over previous diagnostic computer systems (e.g., BUGGY) was its focus on the *common* misconceptions that students experience (e.g., Brown & Burton, 1978; VanLehn, 1982; Young & O'Shea, 1981) rather than all possible misconceptions which could be programmed into a computer.

An innovative feature of TORUS is that its diagnostic system involves two testing sessions instead of one as in the BUGGY system. The first test is group administered and has 40 problems. Coefficient alpha reliability for the

test is .97 and alternate test reliability is .93. Using information garnered from this initial test, TORUS makes an hypothesis about a student's error misconception and constructs a second test to confirm or disconfirm the hypothesis. That is, TORUS presents some problems the student should get right and others the student should miss if its diagnosis is correct.

Cognitive Assessment of Math Errors

Individual student errors in math are highly consistent and follow predictable patterns that indicate misinformed algorithms (Brown & Burton, 1978; Pellegrino & Goldman, 1987; VanLehn, 1983, 1990; Young & O'Shea, 1981). Students who made errors on subtraction tended to create their own incorrect interpretations of algorithms taught by the teacher (Brown & Burton, 1978).

Howard and Woodward (1990), in a study of students with learning disabilities, found that students with similar scores on math work samples had different error patterns. Also, students can miss similar problem types for different reasons. Teachers must understand why student errors occur before appropriate remediation can follow.

Diagnostic assessment that provides information about why errors are made gives teachers some understanding of student thinking processes and should lead to better remediation instruction.

For example, if the following problems were missed on a student worksheet,

306	470	7463	490	6066
- 152	- 115	- 634	- 72	- 963
254	365	7829	422	6903

and these problems were correct,

508	6005	904	870
- 102	- 2016	- 45	- 20
406	3989	859	850

an observant teacher would have some idea that the student's problem involved borrowing with zeros. But even this level of diagnosis is possible only if a teacher takes the time to closely examine student errors distributed on a worksheet.

A curriculum based measurement test may show that the student in the example above is progressing along the aim line and making adequate progress. Because the total percent correct for this student may be high enough to exclude further concern from the teacher, this student's misconception could go unnoticed and unremedied.

An analysis of this student's errors by an expert computer system would find that the errors are systematic and predictable. Every time the student needs to borrow for the first time and there is a zero in the minuend, the student writes the number from the subtrahend in the answer for that column. A computer assessment system would provide this hypothesis along with several additional problems for the student to work in order to confirm the hypothesis. Without having to figure out the error patterns themselves, teachers can spend their time confirming computer analyses of student misconceptions and improving student understanding of math.

The Need for Validation of Expert Computer Systems

Research in cognitive assessment of mathematics would be difficult without the use of technology (Goldman, Pellegrino, & Mertz, 1988; Hasselbring, Goin, & Bransford, 1988). The BUGGY and TORUS research in math assessment has been innovative for both educational technology and assessment procedures. Recent trends in CBM assessment include expert computer systems to help teachers make informed instructional decisions (Fuchs, Allinder, Hamlett, & Fuchs, 1990).

With interest in computer based cognitive assessment growing, there is a need to validate these systems. That is, the accuracy of the computer system's output needs to be examined. Hofmeister (1986) discussed this need and described the validation procedures for their expert system (CLASS.LD) designed to confirm placement decisions for students with learning disabilities. Their validation method was based on computer agreement with "experts" in the field of learning disabilities.

Validation of computer systems must be based on human agreement or disagreement with the output of the system. If a teacher cannot confirm information provided by a computer system then that information is not useful to the teacher. TORUS, the focus of this project, was designed to be used in conjunction with teachers who are able to judge the appropriateness and quality of the information provided by the system.

The Research Questions

What is the reliability of the TORUS system in diagnosing students' performance in subtraction when compared to human expert assessment in the following areas: (a) percent correct (b) missed problem types (c) bugs or systematic misconceptions?

Secondly, how easy were TORUS results to interpret? In other words, how useful were these results under naturalistic settings compared to more traditional means of assessing student performance?

Method

Subjects and Setting

One fourth grade classroom with 26 students was chosen at a rural elementary school to participate in this study. Letters explaining the study and asking for parental consent were sent home with all the students in the

classroom. The 22 students who returned parent consent letters were the final participants in the study.

Materials

Students were given the first phase of the TORUS test, which consisted of 40 subtraction problems representing various problem types and degrees of difficulty. From the results of this assessment, TORUS provided the following information for each student: (a) percent correct (b) missed problem types and (c) an hypothesis about misconceptions or difficult problem types for a student (see Appendix A for an example).

There were two types of hypotheses involving errors: misconceptions (i.e., bugs: predictably wrong answers due to incorrect algorithms) or a problem type errors (i.e., a consistent pattern of wrong answers, but no predictable answer to the problems).

Procedures

The 40 problem TORUS test was group administered to all subjects by the classroom teacher. The test took approximately 20 minutes, and it was given during a regular math period during the school day. This is the total involvement of the students with the project.

The test results were analyzed separately by TORUS and the teacher. A comparison was then conducted for three categories: (a) total percent correct (b) problem types errors and (c) misconceptions. The teacher's results were compared with the TORUS results to determine the amount of agreement and disagreement for the three categories of data. This information was recorded on the Performance Data Comparison Form (see Appendix B).

Finally, the teacher reexamined the test sheets of all students for which there were disagreements between the teacher and TORUS. The purpose of this final step was to determine if the TORUS generated data and hypotheses

could provide information that would cause the teacher to alter his initial diagnosis. This was a critical aspect in determining the usefulness of the TORUS system in a classroom setting. If TORUS could provide a systematic analysis that was initially unseen by the teacher, then it would suggest that TORUS was a more viable classroom assessment tool for diagnosing student misconceptions in subtraction that a teacher performing the same task.

Data Analysis

Analyses performed by TORUS and the expert teacher were examined independently by a researcher familiar with the project. There was a 100 percent agreement for the total percent correct. Agreement between TORUS and the teacher on missed problem types was 93 percent.

Finally, there was an initial agreement of 19 of 22 on misconceptions or bugs. The three forms where there was disagreement were returned to the teacher with the TORUS analyses. After reviewing the TORUS findings, the expert teacher indicated that two of the three provided the same analysis and that the difference was semantic (i.e., he described the misconception in different terms). This changed the agreement level to 95 percent.

Given the importance of agreement on misconceptions to the project, a chi-square analysis was conducted on the two sets of analyses. Results were non-significant ($\chi^2 = .09$; $p = .76$).

The time taken by the teacher to record data for each student will be noted on the Performance Data Comparison Form. An average time taken per student was calculated for both methods of diagnosing student errors being compared in this study: the teacher working alone and the TORUS system.

TORUS required approximately three minutes per student for its analysis. Furthermore, student data were done in batch mode for the entire

class. This time required by the teacher to analyzed 22 students was approximately 15 minutes per student. Furthermore, the work was very demanding, and it had to be done in small sets to insure a high quality analysis. The teacher also examined the TORUS analyses at the end of the project and concluded that its results were clear, concise, and informative.

On the whole, the TORUS method for detecting and tracking student misconceptions was much more efficient and potentially more reliable than a teacher who was well trained in analyzing student worksheets. It was also apparent from this experience that responding to each misconception in isolation was not the best method for instructional decision making. Rather, students with misconceptions needed a deeper, more systematic and conceptual approach to the subject than they traditional get in math classes.

Appendix A

Sample of a TORUS Student Profile

Student: Date:

Performance Data:

	Number of Problems	Number Correct	Percent Correct
Non-borrow problems	5	4	80
Borrow problems	25	15	60
Single non zero borrows	5	5	100
Multiple non zero borrows	5	4	80
Single zero borrows	5	2	40
Multiple zero borrows	5	1	20
Multiple mixed borrows	5	3	60
TOTAL TEST	30	19	63

Error Pattern:

The student has difficulties working subtraction problems when there is a borrow for a zero. On the test, this student missed 90 percent of the problems that looked like this:

902	1008	680	7084	403
- 131	- 64	- 7	- 846	- 42
831	1064	687	7838	441

When there is a problem that requires borrowing for the zero, the student writes the bottom number in the answer (e.g., 0 -9 = 9, 0 - 4 = 4) and then goes to the next column.

A TORUS hypothesis of a student's errors may target a problem type or a misconception. A problem type hypothesis designates a type of problem a student misses but doesn't explain what the student is doing wrong. The only predictable aspect is that the student tends to miss that type of problem.

In contrast, an hypothesis of a misconception explains what the student does wrong to miss the problem. So a student's misconception is predictable in how the student will miss the problem. Below are examples of misconceptions and problem types.

Problem Types

There are six mutually exclusive problem types identified by the designers of TORUS. The following list includes examples of the problem types.

1. non-borrow:
   637
   - 516

2. single non-zero borrow:
   648
   - 375

3. multiple non-zero borrow:
   9438
   - 289

4. single zero borrow:
   6304
   - 1154

5. multiple zero borrow:
   4050
   - 3419

6. multiple mixed borrow:
   4088
   - 479

Another problem type that may include any of the six categories above is asymmetrical problems. These are problems with a different number of digits in the minuend and the subtrahend.

Misconceptions

There are many misconceptions that the program can diagnose. These are incorrect algorithms that students use every time a particular situation occurs in a problem. Zeros in problems account for many student errors in subtraction.

Example: Borrowing for a zero bug.

This error pattern exists when a student incorrectly borrows from a zero that hasn't been altered due to previous borrowing. For example,

804	70	6040	270
- 371	- 46	- 657	- 194
573	36	5397	84

these problems would be missed by the student in a predictable fashion. The student might just put the number below the zero in the answer column (0-N = N). Notice that in the third problem above the student would put a 7 in the ones column of the answer (incorrect) but a 3 in the hundreds column (correct) because the student can change zeros to a 9 when necessary. Not being able to change zeros to nines when necessary is a different bug (borrowing from a zero bug).

Another way a student could show this bug is by just putting a zero in the answer column (0-N = 0). Or a student might change all zeros to nines even when a zero is being borrowed from for the first time and should be made into a 10.

Regardless of the predictable way the student missed the problems, the above bug would be confirmed by the second TORUS individualized test using some minimally different problems like the ones below.

5007	708	7037	306
- 328	- 469	- 754	- 145
4679	239	6283	241

For a student who used a 0-N=N strategy for the borrow for a zero bug, TORUS would predict that the student would get the first 3 problems correct and miss the fourth problem.

Many times students have multiple bugs. For example, every time there was a zero in the minuend a student could respond in a predictable incorrect way. If a student always gave up when there was a zero in the minuend that would be a problem type hypothesis rather than a bug. For a misconception to exist a student makes predictable and systematic errors as a result of employing an incorrect strategy (i.e., algorithm).

Appendix B

Performance Data Comparison Form

Student: _____ Date:

Time Taken: _____

	TORUS	TEACHER (researcher)
1. Items Correct:	---/40 = %	---/40 = %
2. Problem Types:	missed/total	missed/total
Non-borrows:	_____	_____
Single non-zero borrows:	_____	_____
Multiple non-zero borrows:	_____	_____
Single zero borrows:	_____	_____
Multiple zero borrows:	_____	_____
Multiple mixed borrows:	_____	_____
3. Error Pattern:		

Teacher Change Form After Looking at TORUS Results:

Student: _____ Date: _____

TORUS hypothesis of student error pattern:

Teacher hypothesis of student error pattern:

Teacher Reason For Change:

References

Algozzine, B., & Ysseldyke, J.E. (1982). Critical issues in special and remedial education. Boston: Houghton Mifflin.

Brown, A.L. & Campione, J. C. (1986). Psychological theory and the study of learning disabilities. American Psychologist, 14(10), 1059-1068.

Brown, J. S., & Burton, R. (1978). Diagnostic models for procedural bugs in basic mathematic skills. Cognitive Science, 2, 155-168.

Carpenter, T.P., Moser, J.M. & Romberg, T.A. (1982), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cawley, J.F. (1985). Cognition and the learning disabled. In J.F. Cawley (Ed.), Cognitive strategies and mathematics for the learning disabled (pp. 1-29). Rockville, MD: Aspen Systems Corporation.

Erlwanger, S. H. (1973). Benny's conception of rules and answers in IPI Mathematics. Journal of Children's Mathematical Behavior, 1, 7-26.

Davis, R.B. (1983). Complex mathematical cognition. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Ferrara, R.A. (1987). Learning mathematics in the zone of proximal development: The importance of flexible use of knowledge. Unpublished doctoral dissertation. University of Illinois at Urbana-Champaign.

Fuchs, L., Allinder, R., Hamlett, C., & Fuchs, D. (1990). An analysis of spelling curricula and teachers' skills in identifying error types. Reading and Special Education, 11(1), 42-52.

Fuchs, L., & Fuchs, D. (1986). Effects of systematic formative evaluation: A meta-analysis. Exceptional Children, 53(3), 199-208.

Fuchs, L. & Hamlett, C. (1988, December). Design and implementation of an expert system: developing effective instructional programs with curriculum-based assessment. Paper presented at the CEC/TAM Conference on Special Education and Technology, Reno, NV.

Ginsburg, H.P., Kossan, N.E., Schwartz, R., & Swanson, D. (1983). Protocol methods in research on mathematical thinking. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Gersten, R., & Kelly, B. (1991). Coaching teachers in the effective implementation of videodisc instruction: Four case studies. Submitted for Publication.

Gersten, R., Woodward, J., & Darch, C. (1986). Direct instruction: A research-based approach to curriculum design and teaching. Exceptional Children, 53, 1, 17-31.

Ginsburg, H. P. (1983). The development of mathematical thinking. Orlando, Florida: Academic Press.

Ginsburg, H.P. (1987). Assessing arithmetic. In D. Hammill (Ed.), Assessing the abilities and instructional needs of students. Austin, TX: ProEd.

Glass, G. V. (1983). Effectiveness of special education. Policy Studies Review, 2(1), 65-78.

Goldman, S. R., Pellegrino, J. W., & Mertz, D. L. (1988). Extended practice of basic addition facts: Strategy changes in learning disabled students. Cognition and Instruction, 5, 223-265.

Good, R. H., & Salvia, J. (1988). Curriculum bias in published, norm-referenced reading tests: Demonstrable effects. School Psychology Review, 17(1), 51-60.

Hammill, D.D. (1987). Assessing the abilities and instructional needs of students. Austin, TX: ProEd.

Hasselbring, T.S., Goin, L. I., & Bransford, J. D. (1988). Developing math automaticity in learning handicapped children: The role of computerized drill and practice. Focus on Exceptional Children, 20(6), 1-7.

Hofmeister, A. (1986). Formative evaluation in the development and validation of expert systems in education. Computational Intelligence, 2(2), 65-67.

Howard, L., & Woodward, J. P. (1990). Misconceptions in subtraction: An analysis of secondary learning disabled students. (Tech. Rep. No. 90-1). Eugene, Oregon: Eugene Research Institute.

Idol, L. (1986). Models of curriculum-based assessment. Rockville, MD: Aspen Systems.

Jenkins, J. R., & Pany, D. (1978). Curriculum biases in reading achievement. Journal of Reading Behavior, 10(4), 345-357.

Leinhardt, G. & Smith, D. (1985). Expertise in mathematics knowledge: Subject matter knowledge. Journal of Educational Psychology, 77(3), 247-271.

McLeod, T., & Armstrong, S. (1982). Learning disabilities in mathematics – Skill deficits and remedial approaches at the intermediate and secondary level. Learning Disability Quarterly, 5, 305-311.

Pellegrino, J.W., & Goldman, S.J. (1987). Information processing and elementary mathematics. Journal of Learning Disabilities, 20, 23-32.

Resnick, L.B., & Ford, W.W. (1981). The psychology of mathematics for instruction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Salvia, J., & Ysseldyke,J. (1985). Assessment in special and remedial education (3rd ed.). Boston: Houghton-Mifflin.

Silbert, J., Carnine, D., & Stein, M. (1981). Direct instruction mathematics. Columbus, OH: Charles Merrill.

Valencia, S. W. & Pearson, P. D. (1988). Principles for classroom comprehension assessment. Remedial and Special Education, 9(1), 26-35.

VanLehn, K. (1983). On the representation of procedures in repair theory. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

VanLehn, K. (1990). Mind bugs. Cambridge, MA: MIT Press.

Woodward, J. (in press). Procedural knowledge in mathematics: The role of curriculum. Journal of Learning Disabilities.

Woodward, J., & Carnine, D. (1988). Antecedent knowledge and intelligent computer assisted instruction. Journal of Learning Disabilities, 21(3), 131-39.

Woodward, J., Freeman, S., Blake, G., & Howard (1990). TORUS: Computer-based analysis of subtraction. [computer program]. Eugene, Oregon: Eugene Research Institute.

Woodward, J., & Gersten, R. (in press). Innovative technology for secondary learning disabled students: A multi-faceted study of implementation. Exceptional Children.

Woodward, J., & Noell, J. (1991). Science instruction at the secondary level: Implications for students with learning disabilities. Journal of Learning Disabilities, 24(5), 277-284.

Young, R.M. & O'Shea, T. (1981). Errors in children's subtraction. Cognitive Science, 5, 152-177.

Ysseldyke, J. E., O'Sullivan, P. J., Thurlow, M. L., & Christenson, S. I. (1989). Qualitative differences in reading and math instruction received by handicapped students. Remedial and Special Education, 10(1), 14-20.

4.0 Software Development in Special Education

Software Development in Special Education

John Woodward

Eugene Research Institute

John Noell

Oregon Research Institute

submitted for publication

Abstract

Over the last ten years, special educators have developed a variety of software programs which attempt to address the unique needs of mildly handicapped students. Much of this work has been funded by the Division of Innovation and Development in the Office of Special Education Programs. This article examines software development techniques used by many these special educators through the perspective of the software lifecycle. This perspective, which is used intimately by professional software developers, is rarely discussed in the special education technology literature. With rapid advancements in computers, particularly the graphical user interface, the authors argue that traditional programming techniques are becoming less and less viable for software development within the typical funding levels and timelines of federal initiatives. Instead, a trend toward the use of commercial software tools is viewed as inevitable given these constraints. The article concludes with a brief description of two programs recently completed in the area of advanced uses of technology in assessment.

For over ten years, the Division of Innovation and Development (DID) in the U.S. Department of Education's Office of Special Education Programs has supported a wide range of initiatives designed to enhance the state of technology use for handicapped individuals. Technology researchers and developers have studied the effectiveness of computer-assisted instructional programs for special education students; probed potential applications of technology currently used in business and the military for the handicapped population; developed an array of compensatory applications for physical, sensory, or cognitive impairments; and designed innovative instructional, assessment, and multimedia systems. There have also been an untold number of other computer-based research and development activities in special education which have not been funded by DID-directed technology competitions.

A sustaining impetus for these activities comes from survey and observational studies conducted during the early 1980's. Researchers found that elementary and secondary mildly handicapped students spend a disproportionate amount of time on drill and practice programs (Becker & Sterling, 1987; Cosden, Gerber, Semmel, Goldman, & Semmel, 1987; Rieth, Bahr, Okolo, Polsgrove, & Eckert, 1988; Semmel & Lieber, 1986). Where technology was not used for drill and practice, the main intent of computer use with special education students was to improve motivation, self-confidence, and self-discipline.

A subtle, but more germane finding in these studies was the modest variation in the *type* of software used (Cosden et al., 1987). It was not uncommon to find students of all academic abilities practicing the same math facts, vocabulary, or decoding software programs. Rieth et al. (1988) attributed part of the infrequent use of microcomputers at the secondary special education level to the lack of diverse, and hence, appropriate software.

In a broad sense, commercial software commonly available to special education students has failed to address their individual academic needs. Federally sponsored projects, when viewed in this light, have provided alternatives typically unavailable from large software publishing companies. Some special education software programs developed under these monies have reached the commercial market, while others have remained in a prototypic form. Nonetheless, a common theme pervading special education software projects over the last decade has been the incorporation of assessment (Deno, 1990; Fuchs, Fuchs, Hamlett, & Stecker, 1991) or instructional design (Carnine, 1989; Hofmeister, 1990) features which address the *unique* needs of handicapped students.

This article discusses the software tools commonly used to develop computer-based programs for special education students -- specifically programs developed for assessment and instruction of mildly handicapped students. The limits of DID support (both in actual dollars and funding timelines), the paucity of commercial developers for the special education market, and the complexity of today's computers make the selection of appropriate software development tools a fundamental issue for the special education technologist. This observation remains valid even if programs are only developed as prototypes. Often the decisions as to which tools are best suited for successful software development are as complex as the assessment and/or instructional design techniques to be embedded in the program. This article concludes with a brief description of two currently funded DID technology projects which, in one case uses tools that until recently were only available in business and the military.

Tools for Software Development

Programming Languages

Programming languages such as BASIC, FORTRAN, and Pascal are the most traditional method of developing educational software programs. These "high level" languages enable the programmer to manipulate the basic operations of a computer in a more comprehensible manner than binary or assembly language code. High level languages, then, represent an important historical step in expediting the development of a software program. They move the programmer closer to the "problem space" (i.e., what software and curriculum developers want a program to do), and further away from the underlying machine hardware. The logic of making programming environments increasingly comprehensible can be extended to even higher level languages such as authoring languages, authoring systems, and expert systems, all of which will be discussed later.

First and second generation high level languages such as FORTRAN and BASIC have been particularly useful vehicles for novice computing. They have provided an economical, iterative means of processing large sets of data for relatively simple programs. However, computing in the 1970's and, most certainly in the 1980's, shifted toward more structured programming languages such as Pascal, and C because of their modular structure and transportability across computer platforms. Object-oriented programming is the most recent advancement in high level langauges for enhancing modularity.

Writing instructional and assessment programs in a traditional, high level language -- a path followed by many special education technologists -- has obvious advantages. It facilitates the development of programs which are closest to educational specifications. CAI programs in math facts (Goldman, Pellegrino, & Mertz, 1988; Hasselbring, Goin, & Bransford, 1988), spelling (Fuchs, Hamlett, &

Fuchs, 1990; Hasselbring, 1982), sight words (Lally, 1981), vocabulary (Carnine, Granzin, & Rankin, 1983; Johnson, Gersten, & Carnine, 1987), health education (Carnine, Lang, & Wong, 1983; Woodward, Carnine, & Gersten, 1988) and reasoning skills (Collins, Carnine, & Gersten, 1987; Engelmann & Carnine, 1983) are but some examples of successfully developed and researched special education software. Typically, these programs contain elaborate branching schemes and conditional forms of feedback.

Comparable development work has also occurred in computer-managed instruction. Computer-based implementations of curriculum-based measurement (CBM; Fuchs, Deno, & Mirkin, 1982; Fuchs, Fuchs, Hamlett, & Hasselbring, 1987; Germann, 1985) have significantly reduced its labor intensive aspects. Data management software that automatically graphs data, applies decision rules, and provides feedback statements summarizing those decisions have led to positive teacher attitudes about CBM as an efficient means of formative evaluation (Fuchs et al., 1987).

The Graphic Interface and Increased Programming Complexity

Software development through higher level languages has been feasible insofar as programs were not excessively complex, and they did not demand an extensive use of graphics. However, software trends over the last two decades have been in the direction of greater complexity and graphics as hardware costs have rapidly declined. Graphical user interfaces (GUIs) such as the Macintosh operating system and Microsoft Windows exemplify this movement.

These environments demand a sophisticated understanding of the machine's operating system. They require the use of ancillary programming tools (e.g., MPW for the Macintosh) with long learning curves, thus necessitating a great commitment of time and effort. With few exceptions, creating programs

in these environments requires one or more experienced professional microcomputer programmers.

Few programs have appeared in the special education literature which are written for GUI environments (Gleason, Carnine, & Vala, 1991; Slocum, 1988; Woodward & Carnine, 1989; Woodward, Carnine, Steely, Freeman, & Nospitz, 1988). The demands of these environments highlight the importance of the software lifecycle as a framework for conceptualizing software development, a topic rarely discussed in the special education technology literature.

The Software Lifecycle

Most successful software is developed, disseminated, and revised using a multi-step procedure. These steps have been used as common operating procedures in commercial development environments for decades, and they comprise an implicit framework for professional developers. However, the use of a software lifecycle in special education research and development has been far less systematic. Figure 1 below presents the basic steps or phases of the software lifecycle.

[insert Figure 1 about here]

Program objectives. The first step is to define the purpose of the software, choose which hardware platform is most suitable, and select appropriate software tools (e.g., a high level language such as C, an expert system shell, hypermedia). For the special educator, this phase requires extensive planning and specification. Unlike print curricula development, for example, which can be revised in an interactive process encompassing isolated portions of the materials, software programming initially requires a much more complete initial layout of the problem space (i.e., exactly what a program should do and when different events should be occur). It is rarely possible to "try out" small portions of a program. Educators who decide to make extensive changes in the way a program works

after "tryouts" force extensive changes in the entire computer program, which can severely delay or hamper the software development process.

Data flow design. The second step in the software lifecycle is the design or layout of the data flow. Programmers have traditionally used data flow diagrams or flow charts to accomplish this purpose. The sequence of events (e.g., how and what data flow from one step to the next), the branching criteria, and so forth are fully specified. Data flow diagrams are largely historical methods associated with older, more traditional languages (e.g., Pascal). They may be less useful when applied to artificial intelligence languages.

Newer tools built around graphic environments enable the programmer to rapidly generate "shells" of the different screens in a program, much in the way an advertising agency uses a story board in developing television commercials. These shells, which are critical to educational software development, only display the format of what a user will see at each point in the program and have very little code associated with them. Also, no code links one screen to another in the program (i.e., there is no way to "run" the program at this point). However, this aspect of the software lifecycle is crucial to the final product because it previews the program's interface. This is the first major opportunity for developers to reach consensus on the "look and feel" of the program before the intense coding phase begins. The flow of information between user and computer program is at least as important as the internal flow of data within the program.

Coding. After the overall design is finalized, the third step is the actual coding or computer programming. The first part of this step is to lay out the design of the code, which has evolved from the preceding steps. Segments of the program are written, and in the case of larger programs, the work is subdivided so that several teams can work on the project simultaneously. As portions of the code are written, they are checked for bugs (e.g., typos, syntax errors, logic errors).

Although many educators tend to think that coding is the last phase of the software development process, it is part of an iterative process and there is much left to be done.

Alpha and beta testing. The program, once written and assembled, is now available for the first level of testing, called "alpha" testing. As with any complex task involving the creation of hundreds, if not thousands or tens of thousands of lines of code, there are bound to be mistakes. Programmers search for bugs at a larger scale, such as errors in the overall design or subcomponents of the program that don't work together. One of the ultimate goals of this process is to insure that the program is robust (i.e., that it performs faultlessly under a variety of uses and conditions). Depending on the complexity of the program and other variables, repeated episodes of alpha testing followed by additional programming may be necessary.

Once the alpha testing and re-programming phase is completed, programs are usually sent to actual end-users for a further period of testing called "beta testing." Again, almost always, actual end-users will encounter problems not foreseen or adequately dealt with in earlier phases. This stage also provides a final opportunity to judge whether or not the developer's intentions match those of a wide range of users. Mismatches or "intent bugs" mean that the program must be modified to satisfy the demands of the eventual user. Programs that analyze a complex array of student data but are difficult to use because data entry is confusing is one example of an intent bug.

Additional programming is done following beta testing to rectify all problems identified to date. The greater the complexity of the program, the longer each of the preceding phases takes. For special educators, this is a difficult task because there is generally not a widely installed base of users who can sufficiently test the robustness of a program.

Marketing, dissemination, and technical support. The program is now ready for dissemination. Some developers sell their products directly, while others -- particularly educators -- use publishers or distributors. Although more of the purchase price is retained with direct sales, the time, money, and expertise required for successful marketing may make the publisher/ distributor option more desirable. A critical element is the need for technical support. Obviously, the more complex the program, the more likely that end-users will want some kind of technical support. This is a regular feature of large software programs, something which is rarely provided for in special educational software.

Program updating. Over time, programs are continually updated and, in some cases, rewritten entirely. This pattern of development and refinement is evident in the CBM software (Fuchs et al., 1982; Fuchs et al., 1991). Development platforms have shifted from those written in high level languages to ones based on expert system shells.

The process of developing software is usually complicated, time-consuming, expensive, and the magnitude of the task being contemplated is often underestimated. Companies that are in the business of creating software often have very large teams of programmers who spend years refining a product. Development costs can easily reach into hundreds of thousands or millions of dollars. With large potential markets, such costs may be recouped. In special education, the market is generally quite small and software tends to lack a broad-based appeal so that the amortization of large development costs is not a realistic goal. This problem is exacerbated when programs are developed with federal dollars which are allotted over strictly controlled time periods. Figure 2 portrays the problem of special education software development in the context of professional software development and its lifecycle.

[Insert Figure 2 about here]

Courseware Authoring Environments

It should be evident that creating programs with high level languages requires a considerable amount of pre-planning, expertise, and iterative development *before* they are ever used on classrooms. The cost and timelines associated with this kind of development have led educators to look at authoring languages and systems as serious alternatives for producing simple drill and practice and tutorial programs (Woodward & Carnine, 1983).

Authoring languages. Authoring languages have been advertised as quick solutions to CAI for educators who are intimidated by high-level languages such as BASIC, Pascal, or C. Languages such as *Pilot* (e.g., Apple *Pilot* and Atari *Pilot*) provide a simplified code for creating modules of instruction. Their "code" or set of commands is simpler and supposedly more intuitive, which means that it is more readily understandable to the casual or novice programmer. For example, "T:" signifies "text" and would be comparable to PRINT in BASIC or WRITELN in Pascal. With this command, one can write statements (or even leave blank lines) to the screen for a presentation. With "A:" the program will "accept" responses from the student. Such answers can be compared and judged correct by an "M:" or match command.

Authoring languages have a distinct appeal for educators and instructional designers. Freed from the minute concerns of traditional programming -- particularly intensive data flow design -- users presumably can concentrate more on the program objectives phase of the development cycle. Segments of code can be strung together in more intuitive, linear segments.

However, a closer look at this courseware design process indicates that the programming phase of the software lifecycle remains a critical issue. What one trades in using an authoring language such as Pilot (as opposed to Pascal, for example) is power for relative ease of use. As Merrill (1982) notes, "the more

powerful languages offer the advantage of additional capabilities when the author is ready to go beyond the minimal subset. If an author begins by learning *Pilot* and then desires greater power, he must scrap *Pilot* and begin by learning a new language. Why not begin with a powerful language in the first place?" (p. 76).

The simplified nature of commercial authoring languages leads to further problems. There is nothing in these systems that require the developer to use a structured or modular approach to programming. For many courseware authors, this may encourage mediocre programming strategies (Merrill, 1982). Furthermore, authoring languages still demand a significant commitment of time for the design of the lesson and its programming.

One study (Woodward & Carnine, 1983) of special educators who were interested in courseware development (but were programming novices) indicated that they were frustrated and often confused by the format and syntax of the popular, "*Pilot*-like" authoring languages when they were directed to develop a simple drill and practice program with feedback. While some of the confusion would subside with increased use and familiarity, the person-hour investment in programming remains. Recent research (Rode & Poirot, 1989) indicates that teachers trained to use authoring languages, especially those who teach non-computer related subjects, do not continue to write software programs for their everyday instruction.

These criticisms suggest that the time educators hope to save in the coding phase of software development -- at least for lengthy programs -- is misleading. Problems arising in the data flow and coding steps of the project may actually lengthen the time require to achieve a beta version of the program.

Authoring systems. A logical alternative for courseware development by programming novices is the use of authoring systems. These systems allow CAI developers to create higher quality courseware in a shorter period of time than

authoring languages or traditional programming languages (Graham, 1983). The user simply answers prompts or questions which are generated by the system. By selecting from a menu that appears in the beginning of the system, a user may include, for example, a graphics or sound component in a lesson. The authoring system programs and formats the input from the user. These systems, then, allow users to create respectable courseware using a specific template.

In this sense, authoring systems solve some of the problems of courseware development where instructional tasks are highly similar (e.g., math facts, lists of spelling words, vocabulary drills). All too often traditional approaches to authoring courseware frequently lead the user to perceive the learning tasks as unique, thus missing "the opportunity to use standard formats" (Brady & Kincaid, 1981-82, p. 117).

Yet as much as authoring systems might save developers time in the program objectives, data flow, and coding phases, critics are quick to note the inherent constraints of this approach to courseware development. "Users must recognize the limits of the template, and not try to force all instruction to fit the template. The use of templates would be more acceptable if a variety of templates were provided for the different types of learning" (Merrill, 1982, p. 77). Thus, what appears to be substantial time and financial savings during the initial phases of the software lifecycle are not realized because the educator is unable to accurately "fit" his or her design into the existing system.

This issue is further complicated when developers construct authoring systems with specific instructional design principles in mind. DIAL (Engelmann et al., 1983), for example, was designed to include a variety of options for feedback, hints, timed responses, and the recycling of missed items, and the system has been used successfully in instructional studies involving mildly handicapped students (Gleason, Carnine, & Boriero, 1989; Grossen & Carnine, 1990).

However, the precise nature of such a system even further constrains alternate instructional strategies for courseware development. Critics have labeled (and dismissed) DIAL-like systems as "electronically programmed textbooks" whose pre-formatted, linear orientation offer very little in the way of sophisticated, "intelligent" instruction (Carbonell, 1970; Merrill, 1987). This characterization is unduly harsh insofar as many academic tasks are best presented through sophisticated drill and practice or tutorial programs. Rather, the criticism raises an issue of "template constraints" which is a persistent concern for a variety of authoring systems (e.g., hypermedia, expert systems).

Newer Tools for Courseware and Assessment Programs

With the growing popularity of GUI systems in industry and education, there has been a commensurate shift in the kinds of tools used to develop instructional and assessment systems in special education. As stated earlier, the GUI environment is exceedingly complex. For GUI-based special education programs to take shape at a reasonable cost and within reasonable time constraints, more sophisticated tools -- ones that are already developed and commercially marketed -- need to be used.

The advantages of commercial software tools and programming environments is manifold. Commercial versions are often quite robust, having been alpha and beta tested well before their use in designing educational software. Second, they frequently contain explicit links to other software programs or peripheral devices (e.g., hypermedia and its links to CD-ROM or videodiscs). Thus, commercial programs serve as powerful building blocks for complex programs. Recent interest in hypermedia and expert systems in special education bears this out.

Hypermedia. While many authoring languages and systems have been ported over to GUI systems, hypermedia has drawn the most attention. The

concept of hypermedia has existed for almost half of a century (Bush, 1945) even though it has only become a viable notion recently, again due to the rapid advances in computing environments over the last two decades. Graphically oriented computers have extended the hypertext environment to visual data or "hypermaps" (Reynolds & Dansereau, 1990). Because of its recency, there are relatively few documented uses to date of hypermedia in special education. Hasselbring, Goin, and Wissick (1989) have developed hypermedia programs for "anchored" instruction. Students work in a multi-media learning lab where reading, spelling, and writing are computer-based activities, and reading words, for example, are keyed to short videodisc segments through a Hypercard™ interface.

As modern authoring languages, hypermedia-based languages are vulnerable to some of the same issues mentioned earlier. The time allocated for coding hypermedia programs easily can be underestimated. Script-oriented languages such as Hypertalk and Supertalk still require a level of competence that exceeds the novice, and the eventual program necessitates extensive design and debugging. This particularly true when the developer wishes to access external devices such as videodisc players. The lack of built-in connections requires the use of XCMDs and XFCNs (i.e., External Commands and External Functions). XCMDs and XFCNs can be written in any high-level language, such as C or Pascal, and they are called or invoked from within a hypermedia script.

Many users who are accustomed to the style of older authoring languages and systems may be surprised to find that hypermedia programs--at least as they are currently designed--do not easily permit highly structured branching schemes which are based on student performance or other linear characteristics associated with traditional CAI programs. These programs tend to be weak in data management and summative profiles of student performance.

There is also a second, more theoretical problem with hypermedia, one associated with its program objectives. The idea of a non-linear browsing environment for instruction is a radical departure from the highly structured, "CAI" character of early microcomputer authoring languages. Many researchers (e.g., Conklin, 1987; Heller, 1990) question whether or not hypermedia can be used for effective instruction given its main pedagogical assumptions (i.e., incidental learning). In this respect, hypermedia suffers the same criticism as highly structured authoring systems like DIAL (Engelmann et al., 1983) -- as a template for courseware development, it constrains the range of program objectives.

This has been most apparent to users who are accustomed to the style (hence, program objectives) of older authoring languages and systems. Hypermedia programs -- at least as they are currently designed -- do not permit highly structured branching schemes which are based on student performance or other linear characteristics associated with traditional CAI programs. These programs are also weak in data management and summative profiles of student performance.

However, all of this is seemingly less of an issue when hypermedia is used as a direct link or "front-end" to other programs. It can be used as an interface for CD-ROM or videodisc presentations or for commercial databases. It also has considerable promise as a building block for complex assessment or instructional programs.

Expert systems. Expert system technologies have had a broad impact in business and, to a lesser extent, in special education over the last decade (Hofmeister & Ferrara, 1986). Increasingly, expert systems are chosen over structured programming languages such as Pascal and C because of the need for specific types of representational and reasoning structures.

As Figure 3 indicates, expert systems have received substantial use as development tools in special education. Once relatively obscure technologies, expert systems have evolved into a common choice for many computer based applications, especially in the area of categorical classification and assessment.

[insert Figure 3 about here]

Expert systems are highly valued for their "distributed expertise." Once encoded in the software program, expert advice can be applied in a variety of settings where human experts are generally unavailable or too costly. To a large extent, this has been the guiding principle behind the expert systems developed for assessing handicapped eligibility by Hofmeister and his colleagues (Ferrara, Parry, & Lubke, 1985; Parry & Hofmeister, 1986; Prater & Althouse, 1986).

Expert systems also provide a distinct advantage in terms of the software lifecycle. Once the program is designed, the coding phase of the project (i.e., the development of the rules) is relatively brief. Thus, rather than "starting from scratch" and writing an entire program in an artificial intelligence language, such as LISP or Prolog, the developer moves very quickly from the design phase to the alpha and beta test stages. Recent improvements in graphical interfaces for expert systems further enable developers to concentrate on the design and alpha and beta testing phases of a project.

Hofmeister (1986) and others have stressed the importance of the alpha and beta test phases as means of determining agreement between human experts and the computer program. The extent to which alpha and beta testing act as a means of validating principles developed in the design phase (rather than correcting syntax errors and programming logic) underscores an important aspect of expert systems: they are easy to misuse and/or underutilize.

Expert systems, by name, are designed to capture and manipulate expertise on a given topic, and the brief history of these systems indicates a clear shift

toward systems which operate on a narrowly defined knowledge base (Bowerman & Glover, 1988; Chandrasekaran & Mittal, 1984; Hayes-Roth, Waterman, & Lenat, 1983). Thus, expert systems are only as good as the knowledge (e.g., its detail, accuracy) they encode, forcing system designers to closely examine and determine if the domain is well-ordered, rich in information, and amenable to clear rules or heuristics (Alvey, Myers, & Greaves, 1985; Bramer, 1982). This also implies that expert systems are often enhanced when linked to databases.

Unfortunately, the lure of expert systems sometimes has misled developers; the technology has often eclipsed an adequate analysis of the knowledge domain. The considerations made at the design phase, particularly the match between an expert system and a given domain, are critical to its success. In this respect, the limits of expert systems are similar to the template constraints mentioned earlier. The developer may attempt to "fit" a poorly designed or conceptualized educational problem into an expert system shell.

Innovative Programs in Assessment and Technology

The two programs discussed below were developed for a recently funded DID competition in advanced uses of technology in assessment. These programs reflect contemporary trends in software development. The first program, TORUS (Woodward, Freeman, Blake, & Howard, 1990), uses an object oriented expert system in conjunction with a database to analyze misconceptions in subtraction. The Social Skills Assessment Program (Irvin et al., in press), the second program, is a multimedia system, employing a hypermedia front-end linked to a videodisc program.

TORUS

TORUS was designed to build upon earlier mathematical assessment programs, particularly BUGGY (Brown & Burton, 1978; VanLehn, 1982, 1990). The BUGGY project went well beyond earlier work in simply classifying error

patterns (e.g., Ashlock, 1986; Engelhardt, 1977; West, 1971). It demonstrated that even on a seemingly rudimentary task such as subtraction, students *actively constructed* interpretations of what they were taught. A cognitive model of the many deep-seated patterns of misconceptions was coded into BUGGY using the LISP programming language.

BUGGY researchers eventually documented over 3000 possible bugs (Burton, 1981), many of which are extremely unique, if not exotic. The vast of array of bugs, some of which were theoretical in nature, stemmed from algorithms in the original BUGGY program which generated a "best possible fit" explanation of response patterns to computational subtraction problems. Lower level bugs were combined to explain the widest range of problems, often yielding highly idiosyncratic bugs (Brown & Burton, 1978). Of these potential bugs, 157 distinct sets of bugs have been described, with only half of them occurring more than once (VanLehn, 1982).

TORUS developers pursued a different strategy by closely analyzing data from over 200 students with learning disabilities to determine which bugs were empirically based and common enough to merit incorporating in the TORUS program. Almost half of the project's timeline was devoted to the program objectives phase of the software lifecycle. Had the developers then attempted to create the program in a high level language such as C, it never would have been completed, much less alpha tested within the funding period.

Commercial products as "building blocks" for TORUS. To expedite the coding phase of TORUS, the developers chose commercial programs for the Macintosh. Creating movable windows, scroll bars, and pop-down menus for the Macintosh, like any other GUI, requires thousands of lines of code. However, the use of FoxBase™, a widely used commercial database, enabled the developers to quickly create a "Mac-like," menu-driven interface that was tailored to the

TORUS system. Coding in FoxBase™ for this portion of the program took approximately two weeks.

Nexpert Object™ was used as the expert system engine for the program. As an object oriented system, Nexpert Object™ enabled programmers to quickly conceptualize and code rule networks. This expert system also can be linked to commercial databases such as FoxBase™ and programming languages like C. Both of these links were established so that a wide range of users could readily enter data into TORUS and the final results could be processed as database reports. Complete details of the TORUS program appear elsewhere (Woodward, 1992).

The role of alpha and beta testing. A common evaluation of expert systems is found in "real world" testing (Bowerman & Glover, 1988; Hayes-Roth et al., 1983; Hofmeister, 1986). This becomes increasingly feasible only if the coding phase of the program is reasonable and rules can be easily adapted to field-test data. Nexpert Object™, through its object orientation and robust design, facilitated field testing with more than 200 middle school students with learning disabilities (Howard & Woodward, 1990).

In the alpha phases of the program, data from many students were run through the program so that developers could refine TORUS's diagnostic scheme to better match human expert judgment. Eventually, a study of 30 students conducted late in the beta phase of development showed a .90 level of agreement between the human expert and the TORUS program. A second study involving TORUS diagnoses and an experienced special education teacher showed agreement on 21 of 22 cases ($c^2 = .09$; $p = .76$).

While TORUS remains a prototype of an advanced assessment system, it is currently used as a key dependent measure in several research projects. The extent to which TORUS can be implemented in day-to-day classroom settings is a complicated issue. Theoretically, it is not clear if teachers should use TORUS

results to attend to each student's misconception (a.k.a. the "brush fire" approach) or teach mathematics more conceptually and use TORUS results as a broad index of understanding. On a practical level, the runtime versions of Nexpert Object™ and FoxBase™ are relatively expensive. Unless the program is rewritten in a high level language, something well-beyond the financial resources of the current developers, it is unlikely that it will be affordable to end users.

Social Skills Assessment Program

The Social Skills Assessment Program or SSAP (Irvin et al., in press) was developed to enable teachers to rapidly diagnose specific social skills deficits in mildly handicapped students, especially those entering the mainstream. These students often have difficulties in both peer relations and teacher relations. Efficient remediation of such deficits requires a precise diagnosis. For busy teachers and overburdened school systems, it is often difficult, if not totally impractical, to conduct the behavioral assessments and observations necessary to determine the exact circumstances and factors involved with the problems associated with a specific child. The SSAP enables teachers to make such assessments more easily and efficiently.

Three basic skill areas are assessed: peer group entry, responding to teasing and provocation, and compliance with teacher directives. The assessment program utilizes video-based stimuli to present typical school situations such as scenes of children playing or teachers looking directly at the subject (i.e. into the camera). The on-screen figures present situations and statements or requests. A student may make a response by touching the video display, which is equipped with a touch-sensitive screen. The timing and location of touches are analyzed by the computer program which then determines the next situation (i.e., video segment) that should be presented. Quick random access to any segment of

assessment video is made possible through the use of a computer-controlled videodisc player.

Designed to present stimuli that are as ecologically appropriate as possible, the SSAP is unique in being a stand-alone video assessment system. Previous use of video stimuli for social skills assessment (e.g., Dodge, 19xx) relied on the concurrent presence of a trained interviewer. The SSAP presents all necessary instructions for use and branches in response to the subject's input. The ability to function without the presence of an adult interviewer or supervisor was considered essential. The demand characteristics placed on a subject when reporting choices and rationales to an adult observer clearly are not the same as those experienced in naturalistic settings.

Creating a stand-alone assessment system meant it was necessary to simultaneously develop appropriate video stimuli and the computer program to control the video presentations. Because coordination of these activities was essential to meeting tight deadlines and fixed budgets, extreme care was given to the program objectives and data flow design stages of development. Once initial specifications were set, extensive use was made of logic flowcharts in the design of data flow. These flowcharts were continually compared to initial design specifications and iteratively revised. Once data flow design was completed, the initial coding and video production occurred simultaneously.

Due to the unique nature of both the video stimuli and touch-screen/videodisc player hardware, it was clear that some portions of the program would require a large number of alpha and beta-testing iterations, while others would not. Furthermore, it was important that minor changes be easy to accomplish under field conditions, with limited expertise. Therefore, the researchers chose to use SuperCard (a HyperCard clone) and Macintosh computers. These environments allowed for shortening the iterative

development cycles. Shortening the cycles saved time and money; using a simpler environment for simpler tasks avoided the use of expensive professional programmer time for simple changes.

The tasks that could not be accomplished directly with Supertalk, the Supercard"scripting" (i.e., programming) language (such as controlling the videodisc player, gathering student responses from the touch sensitive screens) were written in Pascal, and invoked from the SuperCard script (as XCMDs and XFCNs). This required the expertise of a professional programmer.

The careful separation of the design and coding responsibilities was a key feature of this project. In traditional software development schemes, involving programmers in all phases of design has always been crucial. In contrast, re-designs of the assessment component (involving major rearrangements) were accomplished without re-writing high-level code (with subsequent debugging of the program). Specific start/stop points for the video stimulus segments were changed in the field by non-programmers with minimal training (e.g., elements of the graphic interface were moved to new positions on the screen). The professional programmer's time was reserved for tasks outside the Supertalk domain, or where speed of execution or complexity of data manipulation was critical and hence demanded the power that only a high level language could provide.

As with TORUS, a hybrid approach to software development, making use of commercially available "building blocks", allowed the SSAP to be designed, coded, alpha-tested and beta-tested in a short time with limited expenditures. Expert systems, database programs, and scripting environments that allow access to high-level programs but which do not require all code to be created in the high-level language, represent the future direction for software development under conditions of severely limited time and money.

Summary

Software development in special education is constrained by limited federal funding and the lack of a broad-based commercial market, a condition which is unlikely to change in the near future. Recent federal initiatives have reflected a growing realization that sophisticated software programs cannot be developed under one or two-year funding cycles. Instead, development of prototypes has become the most viable avenue for innovation.

Prototypes, by definition, do not place much emphasis on the latter phases of the software development lifecyle (i.e., marketing, dissemination, and technical support; program updating), allowing the special education technologist to devote much more time to program objectives, data flow design, coding, and alpha and beta testing. One could further argue that to successfully embed instructional design or assessment principles into the prototype, an inordinate proportion of time is concentrated in the program objectives and alpha and beta testing phases. With limited fiscal and/or timeline constraints, commercial software tools such as the ones described in this article are becoming the most viable means of completing federally-funded software projects. These tools function as robust building blocks which move the developer toward a realistic product for the special education population.

References

Alvey, P. L., Myers, C. D., & Greaves, M. F. (1985). An analysis of the problems of augmenting a small expert system. In M. A. Bramer (Ed.) Research and development in expert systems. Cambridge: Cambridge University Press.

Ashlock, R. B. (1986). Error patterns in computation. Columbus, OH: Charles Merrill.

Becker, H., & Sterling, C. (1987). Equity in school computer use: National data and neglected considerations. Journal of Educational Computing Research, 3(3), 289 - 311.

Bowerman, R. G., & Glover, D. E. (1988). Putting expert systems into practice. New York: Van Nostrand Reinhold Company.

Bramer, M. A. (1982). A survey and critical review of expert systems research. In D. Michie (Ed.) Introductory readings in expert systems. New York: Gordon and Breach Science Publishers.

Brown, J.S. & VanLehn, K. (1982). Towards a generative theory of "bugs." In T.P. Carpenter, J.M. Moser, & T.A. Romberg (Eds.), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Burton, R.B. (1981). DEBUGGY: Diagnosis of errors in basic mathematical skills. In D.H. Sleeman & J.S. Brown (Eds.), Intelligent tutoring systems. London: Academic Press.

Burton, R., & Brown, J. (1978). Diagnostic models for procedural bugs in basic mathematic skills. Cognitive Science, 2, 155-168.

Bush, V. (1945). As we may think. Atlantic Monthly, 176(1), 101-108.

Carbonell, J. (1970). AI in CAI: An artificial intelligence approach to computer assisted instruction. IEEE Transactions on Man-Machine Systems, MMS, 11(4), 190-202.

Carnine, D. (1989). Teaching complex content to learning disabled students: The role of technology. Exceptional Children, 55(6), 524-533.

Carnine, D., Granzin, A., & Rankin, G. (1983). Learning vocabulary. [Computer program], Teaching Wares, Eugene, Oregon.

Carnine, D., Lang, D., & Wong, L. (1983). Health ways. [Computer program], Teaching Wares, Eugene, Oregon.

Chandrasekaran, B., & Mittal, S. (1984). Deep versus compiled knowledge approaches to diagnostic problem solving. In M. J. Coombs (Ed.) Developments in expert systems. London: Academic Press.

Collins, M., Carnine, D., & Gersten, R. (1987). Elaborated corrective feedback and the acquisition of reasoning skills: A study of computer-assisted instruction. Exceptional Children, 54 (3), 254-262.

Conklin, J. (1987). Hypertext: An introduction and survey. IEEE Computer, 2(9), 17-41.

Cosden, M. A., Gerber, M. M., Semmel, D. S., Goldman, S. R., & Semmel, M. I. (1987). Microcomputer use within micro-educational environments. Exceptional Children, 53 (5), 399-409.

Deno, S. (1990). Individual differences and individual difference: The essential difference of special education. Journal of Special Education, 24(2), 160-173.

Ellis, E. S., & Sabornie, E. J. (1986). Effective instruction with microcomputers: promises, practices, and preliminary findings. Focus on Exceptional Children, 19 (4), 1-16.

Engelhardt, J. M. (1977). Analysis of children's computational errors: A qualitative approach. British Journal of Educational Psychology, 47, 149-54.

Engelmann, S., & Carnine, D. (1983). Reasoning skills I. [Computer program], Engelmann-Becker Corporation, Eugene, Oregon.

Engelmann, S., Carnine, D., & Weiss, A. (1983) Direct instruction authoring language. [Computer program], Engelmann-Becker Corporation, Eugene, Oregon.

Ferrara, J. M., Parry, J. D., & Lubke, M. M. (1985). Expert systems authoring tools for the microcomputer: Two examples. Educational Technology, April, 1985, 39-41.

Fuchs, L. S., Allinder, R. M., Hamlett, C. L., and Fuchs, D. (1990). An analysis of spelling curricula and teachers' skills in identifying error types. RASE, 11 (1), 42-52.

Fuchs, L. S., Deno, S. L., & Mirkin, P. K. (1982). Data-based program modification: A continuous evaluation system with computer software to facilitate implementation. Journal of Special Education Technology, 6 (2), 50-57.

Fuchs, L. S., Fuchs, D., Hamlett, C. L., & Hasselbring, T. S. (1987). Using computers with curriculum-based monitoring: Effects on teacher efficiency and satisfaction. Journal of Special Education Technology, 8 (4), 14-27.

Fuchs, L., Fuchs, D., Hamlett, C., & Stecker, P. (1991). Effects of curriculum-based measurement and consultation on teacher planning and student achievement in mathematics operations. American Educational Research Journal, 28(3), 617-641.

Germann, G. (1985). SHERI [Unpublished computer program]. Sandstone, MN.

Gleason, M., Carnine, D., & Boriero, D. (1989). Improving CAI effectiveness with attention to instructional design in teaching story problems to mildly handicapped students. Journal of Special Education Technology, 10(3), 129-136.

Gleason, M., Carnine, D., & Vala, N. (1991). Cumulative versus rapid introduction of new information. Exceptional Children, 57(4), 353-358.

Goldman, S. R., Pellegrino, J. W., & Mertz, D. L. (1988). Extended practice of basic addition facts: Strategy changes in learning disabled students. Cognition and Instruction, 5, 223-265.

Graham, N. (1983). The mind tool. St. Paul: West Publishing Company.

Grossen, B., & Carnine, D. (1990). Diagramming a logic strategy: Effects on more difficult problem types and transfer. Learning Disability Quarterly, 13, 168-182.

Hasselbring, T. S. (1982). Remediation spelling problems of learning-handicapped students through the use of microcomputers. Educational Techonology, 22(2), 31-32.

Hasselbring, T. S., Goin, L. I., & Bransford, J. D. (1988). Developing math automaticity in learning handicapped children: The role of computerized drill and practice. Focus on Exceptional Children, 20 (6), 1-7.

Hasselbring, T. S., Goin, L. I., & Wissick, C. (1989). Making knowledge meaningful: Applications of hypermedia. Journal of Special Education Technology, 10 (2), 61-72.

Hayes-Roth, F., Waterman, D., & Lenat, D. (1983). Building expert systems. Reading, MA: Addison-Wesley.

Heller, R. S. (1990). The role of hypermedia in education: A look at the research issues. Journal of Research on Computing in Education, 22 (4), 431-441.

Hofmeister, A. (1986). Formative evaluation in the development and validation of expert systems in education. Computational Intelligence, 2(2), 65-67.

Hofmeister, A. (1990). Individual differences and the form and function of instruction. Journal of Special Education, 24(2), 150-159.

Hofmeister, A. M., & Ferrara, J. M. (1986). Expert systems and special education. Exceptional Children, 53(3), 235-239.

Howard, L., & Woodward, J. P. (1990). Misconceptions in subtraction: An analysis of secondary learning disabled students. (Tech. Rep. No. 90-1). Eugene, Oregon: Eugene Research Institute.

Irvin, L., Wlaker, H., Noell, J., Singer, G., Irvine, B., Marquez, K., & Britz, B. (in press). Measuring children's social skills using microcomputer-based videodisc assessment. Behavior Modification.

Johnson, G., Gersten, R., & Carnine, D. (1987). Effects of instructional design variables on vocabulary acquisition of LD students: A study of computer-assisted instruction. Journal of Learning Disabilities, 20(4), 206-213.

Jones, K. M., Torgesen, J. K., & Sexton, M. A. (1987). Using computer guided practice to increase decoding fluency in learning disabled children: A study using the Hint and Hunt I program. Journal of Learning Disabilities, 20 (2), 122-128.

Lally, M. (1981). Computer-assisted teaching of sight-word recognition for mentally retarded school children. American Journal of Mental Deficiency, 85, 383-388.

Merrill, M. D. (1987). Prescriptions for an authoring system. Journal of Computer-Based Instruction, 14 (1), 1-10.

Nelson, T. (1967). Getting it out of our system. In G. Schecter (Ed.), Information retrieval: A critical view. Washington, D.C.: Thompson Books.

Parry, J. D., & Hofmeister, A. M. (1986). Development and validation of an expert system for special educators. Learning Disability Quarterly, 9(2), 124-132.

Prater, A. & Althouse, B. (1986). LD. Trainer [computer program]. Logan, Utah: Utah State University.

Prater, M. A., & Ferrara, J. M. (1990). Training educators to accurately classify learning disabled students using concept instruction and expert system technology. Journal of Special Education Technology, 10 (3), 147-156.

Reigeluth, C. M. (1979). TICCIT to the future: Advances in instructional theory for CAI. Journal of Computer-Based Instruction, 6 (2), 40-46.

Rieth, H., Bahr, C., Okolo, C., Polsgrove, L., & Eckert, R. (1988). An analysis of the impact of microcomputers on the secondary special education classroom ecology. Journal of Educational Computing Research, 4 (4), 425-441.

Reynolds, S. & Dansereau, D. (1990). The knowledge hypermap: An alternative to hypertext. Computers in Education, 14(5), 409-416.

Roblyer, M. D. (1981). Instructional design versus authoring of courseware: Some crucial differences. AEDS Journal, Summer 1981, 173-181.

Rode, M., & Poirot, J. (1989). Authoring systems--are they used? Journal of Research on Computing in Education, 22 (2), 191-198.

Semmel, M. I., & Lieber, J. A. (1986) Computer applications in instruction. Focus on Exceptional Children, 18 (9), 1-12.

Slocum, T. (1988). IS graphics: Instructional system for graphic facts. [Computer program], Seattle: Experimental Education Unit, University of Washington.

VanLehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. Journal of Mathematical Behavior, 3(2), 3-72.

Walker, H., Irvin, L., Noell, J., & Singer, G. (in press). A construct score approach to the assessment of social compentence: Rationale, technological considerations, and anticipated outcomes. Behavior Modification.

West, T. A. (1971). Diagnosing pupil errors: Looking for patterns. The Arithmetic Teacher, 18, 467-69.

Woodward, J. P., & Carnine, D. W. (1989). The *Genisys* program: Linking content area knowledge to problem solving through technology-based instruction. Journal of Special Education Technology, 10 (2), 99-112.

Woodward, J., & Carnine, D. (1983). What do today's authoring languages and authoring systems mean to today's educators? [Unpublished manuscript], University of Oregon, Eugene.

Woodward, J., Carnine, D., & Gersten, R. (1988). Teaching problem solving through computer simulations. American Educational Research Journal, 25(1), 72-86.

Woodward, J., Carnine, D., Steeley, D., Freeman, S., & Nospitz, C. (1988). Genisys. [Unpublished computer program], Eugene, Oregon.

Woodward, J., Freeman, S., Blake, G., & Howard (1990). TORUS: Computer-based analysis of subtraction. [computer program]. Eugene, Oregon: Eugene Research Institute.

Yin, R. K., & Moore, G. B. (1987). The use of advanced technologies in special education: Prospects from robotics, artificial intelligence, and computer simulation. Journal of Learning Disabilities, 20 (1), 60-63.

Young, R.M. & O'Shea, T. (1981). Errors in children's subtraction. Cognitive Science, 5, 152-177.

Figures

Figure 1

Common Steps in the Software Lifecycle

- **Program Planning and Specification**
- **Data Flow Design**
- **Coding (Computer Programming)**
- **Alpha and Beta Testing**
- **Marketing/ Dissemination and Technical Support**
- **Program Updating**

Figure 2

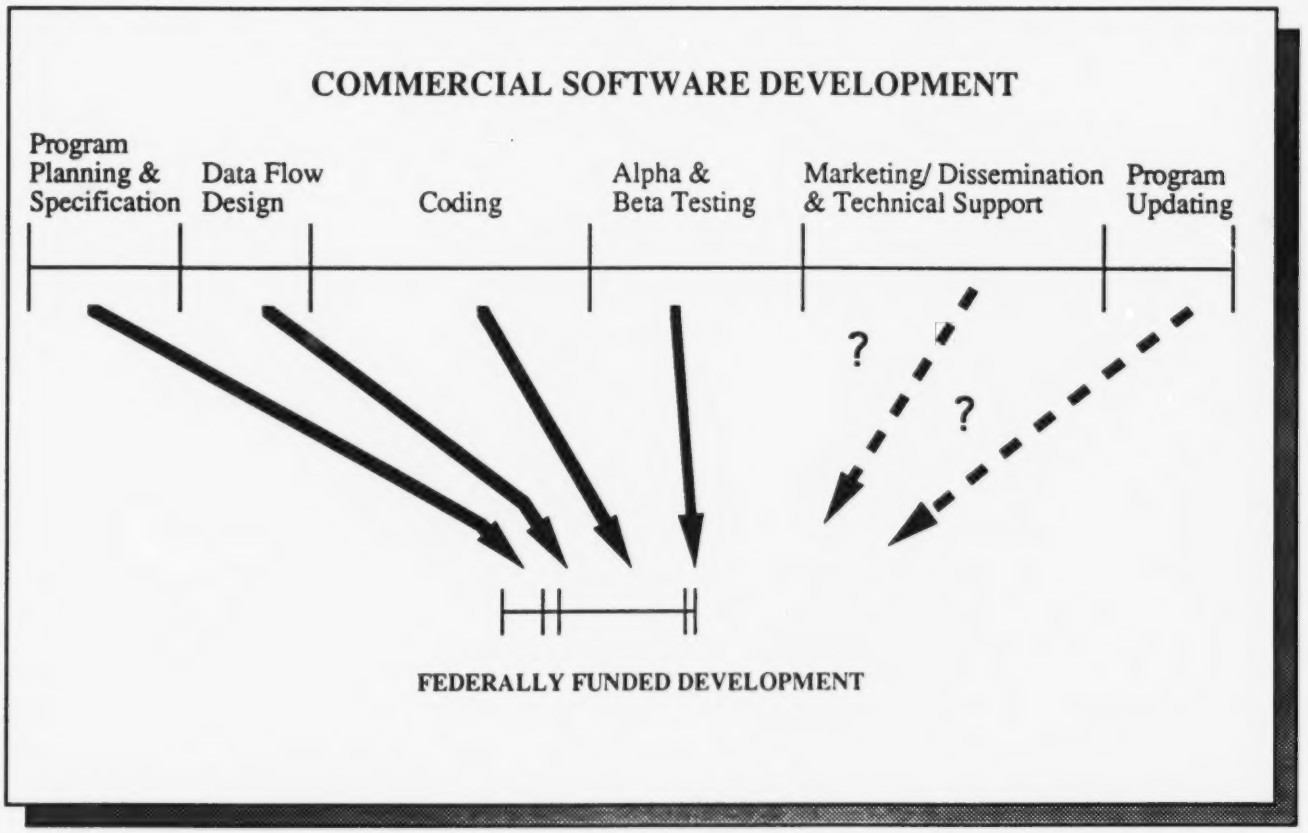

COMMERCIAL SOFTWARE DEVELOPMENT
Program Planning & Specification
Data Flow Design
Coding
Alpha & Beta Testing
Marketing/ Dissemination & Technical Support
Program Updating
?
?
FEDERALLY FUNDED DEVELOPMENT

Figure 3

Expert Systems in Special Education

SYSTEM	AUTHOR(S)	PURPOSE
Math Test Interpreter	Lubke, 1985	diagnosis/ prescription
Class.2	Ferrara, Parry, & Lubke, 1985	classification
Behavior Consultant	Ferrara & Serna, 1985	classification
LD.Trainer	Ferrara & Prater, 1985	classification
SNAP	Haynes & Lubell, 1986	teacher training
Mandate Consultant	Parry & Hofmeister, 1986	regulation compliance
CAPER	Haynes, 1988	planning/ placement
TORUS	Woodward et al., 1990	math assessment
CBM ExSys	Fuchs et al, 1991	assessment/ consultation

5.0 Detailed Comparison of BUGGY and TORUS

The BUGGY Project

The influence of BUGGY Project (J. S. Brown & Burton, 1978; J. S. Brown & Van Lehn, 1982; Burton, 1981; Van Lehn, 1982, 1988) in computer-based education, cognitive psychology, and mathematics education cannot be understated. The project has been cited or discussed repeatedly in key texts on intelligent tutoring systems (Mandl & Lesgold, 1988; Sleeman & J. S. Brown, 1981; Wenger, 1987), mathematics (Carpenter, Moser, & Romberg, 1982; Ginsburg, 1983; Resnick & Ford, 1981; Schoenfeld, 1985) and in innumerable articles on cognition, instruction, and assessment. Its influence equally extends to special education, either as a model for technology development (e.g., Hofmeister & Lubke, 1986; Jones, 1984; Roberts, 1984) or instruction and assessment (e.g., Cawley, 1985; Ferrara, 1987; Goldman, 1989; Pellegrino & Goldman, 1987; Woodward & Carnine, 1988).

Throughout the project and the evolving versions of the BUGGY program (i.e., BUGGY, DEBUGGY, IDEBUGGY), researchers attempted to fully map the specific misconceptions that might arise in a student as he or she learned subtraction. Researchers attempted to apply a cognitive model that explained a wide range of error patterns. In this sense, the BUGGY project went well beyond earlier work in simply classifying error patterns (e.g., Ashlock, 1976; Cox, 1975; Engelhardt, 1977; Harvey & Kyte, 1965; Roberts, 1968; West, 1971). Figure 1 below is a sample of the kinds of errors a student might make when solving a range of subtraction problems. Problems that are incorrectly answered are in circled. These problems, taken from Van Lehn (1982), fully display the difficulty in rendering a systematic analysis across a set of problems. After a very careful analysis, the reader may be able to detect a pattern and proffer one or more hypotheses as to the source of these incorrect answers (i.e., what kind of misconceptions or "bugs" that lead the student to answer problems in a consistent, albeit incorrect, manner).

Figure 1

306	80	183	702	3005	7002	34	251
- 138	- 4	- 95	- 11	- 28	- 239	- 14	- 47
78	76	88	591	1087	4873	24	244

Generally, teachers would describe the student's source of error as one having to do with borrowing, particularly when zeros are present (Van Lehn, 1982). At a more precise level, the student systematically misses those problems where he or she must borrow *from a zero* (this occurs in the first, fourth, fifth, and sixth problems). One could use this information – that a student has difficulty borrowing across zeros – to predict not only the kinds of problems that student would miss (e.g., 408 - 219), but even the answers to such problems. This kind of analysis may be within the ken of a well-trained diagnostician whose job is dedicated to error detection. It is, however, very far beyond the pragmatic limits that general or special education teachers (or aides) face when grading countless worksheets for as many as 30 students in a class.

BUGGY, *as a unique assessment instrument*, provided a computer-based system that could accomplish such an intricate diagnostic task. The system was able to analyze answers to 30 subtraction problems and detect common as well as highly unique patterns of misconceptions. As a computer-based system, it was able to perform these analyses for hundreds of students at a time. Criteria for a bug was its presence in at least three different problems in a set. Some of these misconceptions or bugs were so rare that even though the generative model behind BUGGY predicted that they existed, six were not found in the research (Brown & Van Lehn, 1982).

Nonetheless, A. L. Brown and Campione (1986) specifically refer to the BUGGY project as the kind of system that could enhance teaching, linking assessment to informed instruction. BUGGY demonstrated that even on seemingly perfunctory tasks

such as an arithmetic operation, students actively constructed interpretations of what they were taught – even in spite of direct instruction that attempted to teach one method consistently. This active construction of alternative algorithms has even been documented when regular and LD students learn math facts (Goldman, Pellegrino, & Mertz, 1988; Resnick & Ford, 1982).

From an instructional point of view, BUGGY underscored the distinction between syntax and semantics in arithmetic. That is, many students – and especially those with learning disabilities -- often work problems superficially, only attending to the surface or syntactic features of the problem without attending to what these manipulations mean at a semantic level. In subtraction, BUGGY documented how students endlessly "push around symbols" with little conceptual understanding of what they were doing (Burton, personal communication, 1990; Doyle, 1988; Collins, J. S. Brown, & Newman, 1989). The consequences of this kind of daily symbol pushing are especially dire for learning disabled students, who -- in mathematics at least – may spend the majority of their school lives completing worksheets on basic operations from addition to fractions.

Limitations of the BUGGY project. As a unique contribution to cognitive science, BUGGY was not solely concerned with educational problems. Rather, BUGGY is often described by its authors as an attempt at generative theory -- a precise, theoretical explanation of bug origins, of what bugs should and shouldn't exist in a procedural domain such as subtraction, and what bugs will exist in a procedural skill yet to be analyzed (J. S. Brown & Van Lehn, 1982).

Eventually, this research led to an exceedingly complex model of how bugs -- be they common or highly unique – drive computational errors, and how some students continually "repair" their mistakes by creating new procedures or bugs when they realize that a certain procedure is not working. This cataloguing of buggy

behavior has led to a description of over 3000 possible bugs (Burton, 1981), many of which are extremely unique, if not exotic. Of these potential bugs, 157 distinct sets of bugs have been described, with only half of them occurring more than once (Van Lehn, 1982). Figure 2 below is a sample of such bugs described by Van Lehn.

Figure 2

Sample of Unusual Bugs from Van Lehn (1982)

BORROW INTO ONE = TEN (When a borrow is caused by a 1, the student changes the 1 into a 10 instead of adding 10 to it. For example, 71-38 = 32).

DECRMENTING THE TOP LEFT IS EQUAL TO 0 OR 1 = EIGHT (When borrowing from 0 or 1, changes the 0 or 1 to 8; does not decrement digit to the left of the 0 or 1. For example, 4013-995 = 3778).

N-N = 9 AND DECREMENT THE NEXT COLUMN (When a column has the same number on the top and the bottom the student writes 9 as the answer and decrements the next column to the left even though borrowing is not necessary. For example, 94-34 = 59).

STUTTER SUBTRACTION (When there are blanks in the bottom number, the student subtracts the leftmost digit of the bottom number in every column that has a blank. For example, 4369 - 22 = 2147).

SUBTRACTION BY COPYING FROM THE TOP AND BOTTOM NUMBERS (The student does not subtract. Instead, he copies digits from the exercise to fill in the answer space. He copies the leftmost digit from the top number and the other digits from the bottom number. He will give answers like this: 648 - 231 = 631).

The unusual character of so many of these kinds of bugs help satisfy the requirements of a generative theory. They helped complete a cognitive map that explained the kinds of possible errors a student might make in completing

subtraction problems. However, there are of little practical use to educators -- either in the field as practitioners or researchers who are attempting to link assessment to improved or informed instructional practices. Attempts by those on the BUGGY project to convey subtraction bugs in the complex form as seen in Figure 2 (Friend & Burton, 1981) were largely unsuccessful (Burton, personal communication, 1990; Van Lehn, 1982).

A second, and perhaps more serious concern with results from the BUGGY project was the fact that analyses of misconceptions or bugs were restricted to the computer system itself. That is, there was little systematic effort in observing instructional patterns in classrooms or interviewing and/or watching students as they worked subtraction problems. *Very little was done to explain the origin of these misconceptions in a classroom context*. Further, there was little if any longitudinal research conducted to find the stability of bugs or how they changed as a function of different instructional methods (Van Lehn, 1988). Again, the main focus of the BUGGY project as a generative psychological theory and not direct educational applications.

Finally, the student data base for the project, though considerable in size, was drawn from a very heterogeneous population of students. An original sample of 1325 tests were culled from fourth, fifth, and sixth grade Nicaraguan students (Brown & Burton, 1978; Van Lehn, 1982). Later, 288 American sixth grade students from an upper class community (Haviland, 1979) were added to the BUGGY data base. When the second version of BUGGY was created (i.e., DEBUGGY), 849 students from south San Francisco participated. In all of these studies, no attempt was made to analyze bug patterns by ability. This shortcoming is important, given the likely hypothesis that remedial and learning disabled students would be more prone to systematic, long term bugs. Based on

these limitations and an array of other design considerations, the TORUS program (Woodward, Freeman, Blake, & Howard, 1990) was designed.

The TORUS Program

The explicit intention of TORUS was to build upon the BUGGY research agenda (accounting for some of the deficiencies discussed above) and to explore those features proposed in the latter phases of the BUGGY project. Figure 3 below summarizes the innovative features of the TORUS program.

Figure 3

Innovative Features of the TORUS Program

- Interactive Problem Generation to Confirm Bug Hypotheses
- Criterion Performance Scores as Well as Bug Diagnoses
- Focus on the Most Common Bugs
- Direct Linkage of Bugs to Instructional Remedies
- Field Tested with Learning Disabled Students
- Microcomputer Based Program

Interactive Problem Generation. One of the most engaging ideas proposed near the end of the BUGGY project was an interactive diagnostic system -- later named IDEBUGGY (Burton, 1981). Rather than operate from a fixed set of subtraction problems, many of which may not have been at the appropriate instructional level for the student, Burton (1981) proposed a system that would continually generate problems based on a set of bug hypotheses. Once the

student completed a screening measure, a program such as IDEBUGGY would construct a set of hypotheses and judiciously create problems that would confirm or disconfirm each bug hypothesis. IDEBUGGY was a prototype of this kind of innovative diagnostic system -- one similar in design to many other artificial intelligence diagnostic systems (Sleeman & Brown, 1981; Wenger, 1988). However, IDEBUGGY was never fully completed.

The TORUS program incorporated interactive problem generation by first testing students on a wide range of subtraction problems (Session 1) and then generating a second set of problems tailored to that student's level of competence and/or potential misconceptions. Session 1 contains 30 items representing the range of subtraction problems that intermediate and middle school learning disabled students commonly face. Once answers to the problem set are entered, TORUS systematically analyzes correct and incorrect problems for bug patterns (e.g., the student is not borrowing, the student does not know how to solve the problem when he or she gets to the left most digit, the student cannot borrow when zeros are present). Based on this extensive analysis, another set of 30 problems are generated (Session 2). Problems replicated those where a possible bug exist (e.g., borrowing from zero in any place but the ones column: 104 - 35) and slightly altered problems that would strengthen the hypothesis (e.g., 124 - 35). If a student's misconceptions are restricted to the presence of zeros, then he or she should miss the first type of problem (104 - 35) and **not** the second type (124 - 35).

Criterion Performance Scores. Students do not always err in systematic ways. Inconsistencies, or what came to be known as "slips" in the BUGGY research (Van Lehn, 1982, 1988), were less predictable errors and emanated from fatigue, carelessness, and other sources of inconsistency. Versions of the BUGGY project (i.e., DEBUGGY) were unable to provide informative information about

the student because the performance could not be diagnosed as buggy. It is for this reason that Van Lehn (1982) and others (e.g., Friend, 1981) suggest that a criterion or subskill-based diagnosis should be added to a BUGGY-type program.

This concern for alternate diagnoses, ones that would be educationally meaningful, was a foremost concern when TORUS was designed. TORUS has the potential of providing at least three levels of diagnosis: (a) bugs (e.g., the student consistently errs when borrowing from a zero) (b) systematic errors on problem types (e.g., problems with more than one borrow), and (c) descriptive analyses (e.g., percent correct on total test, on borrowing problems, on "large" problems -- ones with three or four digits). These different levels of diagnoses will help provide meaningful diagnoses for a wide range of student performance.

Focus on Common Bugs. Designed for a special education environment, TORUS avoided cataloging the arcane or exotic bugs that were part of the BUGGY data base (see Table 1.2.1 above). It was felt, as expressed by Van Lehn (1982) and Burton (personal communication, 1990), that these types of bugs would have little import for educators whose need for diagnostic information is at a much broader level (Woodward & Carnine, 1988). Therefore, the algorithms used to find bugs in TORUS are based on two sources: (a) those documented as the most common in the literature (e.g., Brown & Burton, 1978; Van Lehn, 1982; Young & O'Shea, 1981) and (b) those found in a sample of approximately 90 sixth through eighth grade learning disabled students receiving special education instruction in mathematics (Howard & Woodward, 1990).

This latter analysis of learning disabled students revealed that 53 percent of the students had systematic misconceptions (i.e., bugs) or a predicatable errors in certain types of problems (e.g., borrowing involving zeros). Furthermore, mean performance overall for these students was at a distressingly low level -- 69 percent. In other words, middle school learning disabled students, on average,

still do not perform subtraction problems at a level anywhere near mastery *and* exhibit some kind of pattern of errors in their answers.

Remaining Innovations. To augment the educational utility of TORUS, the three levels of diagnosis described above have accompanying remedies. These remedies accompany the final diagnosis given to the teacher. They are written by Gerald Silbert, one of the leading curriculum developers in mathematics for special education students. The remedies are drawn from a long history of field testing and research with low achieving and learning disabled students (Silbert, Carnine, & Stein, 1989).

The TORUS program has been field tested with fourth, fifth, and sixth grade learning disabled students. Finally, TORUS was designed for the Macintosh II computer. Apple Computer's leadership in the educational market is well-known, and the TORUS program requires only one Macintosh computer to analyze the results of an entire class of students. This, it is felt, is a cost effective approach to microcomputer use in schools. BUGGY research conducted a decade ago was done on mainframe computers. However, advances in computing (hardware and software) allow programs of the complexity of BUGGY to be replicated on platforms such as the Macintosh (Burton, personal communication, 1990).

Documenting Misconceptions More Comprehensively

Computer-based assessment systems such as BUGGY or TORUS can provide only so much information about systematic misconceptions or "buggy" behavior. As mentioned earlier, one limitation of the BUGGY project was that bugs were analyzed solely in the context of the computer program. Little was done to methodically document the origin of these bugs or directly link a change in misconceptions to specific instructional techniques. This is not to say that many teaching techniques such as visual representations (e.g., Resnick, 1983;

Resnick & Ford, 1981; Resnick & Omanson, 1987) or scaffolded discussions (Collins et al., 1989; Ferrara, 1987; Lampert, 1986; Resnick, 1988) have not been deeply influenced by the BUGGY research. Rather, a BUGGY-like program or method has not been used as a dependent measure in assessing the success or failure of different methods.

To comprehensively research misconceptions in addition and subtraction, these activities must be linked. Observational research that includes mathematically-oriented interview and protocol techniques (e.g., Carpenter & Moser, 1982 ; Davis, 1983; Ginsburg, Kossan, Schwarts, & Swanson, 1983) documenting the origins and persistence of bugs for low achieving and learning disabled children are critical. Further, experimental methods comparing different instructional interventions, ones that use a system such as TORUS as a main dependent measure, are equally critical. In this manner, the broad issue of misconceptions – their origins, persistence, and remedy -- can be successfully researched.

REFERENCES

Ashlock, R. B. (1976). Error patterns in computation. Columbus, OH: Charles Merrill.

Bowerman, R. G., & Glover, D. E. (1988). Putting expert systems into practice. New York: Van Nostrand Reinhold Company.

Brown, A.L. & Campione, J. C. (1986). Psychological theory and the study of learning disabilities. American Psychologist, 14(10), 1059-1068.

Brown, J. S., & Burton, R. (1978). Diagnostic models for procedural bugs in basic mathematic skills. Cognitive Science, 2, 155-168.

Brown, J.S. & Van Lehn, K. (1982). Towards a generative theory of "bugs." In T.P. Carpenter, J.M. Moser, & T.A. Romberg (Eds.), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Burton, R.B. (1981). DEBUGGY: Diagnosis of errors in basic mathematical skills. In D.H. Sleeman & J.S. Brown (Eds.), Intelligent tutoring systems. London: Academic Press.

Carpenter, T.P. & Moser, J.M. (1982). The development of addition and subtraction problem-solving skills. In T.P. Carpenter, J.M. Moser, & T.A. Romberg (Eds.), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Carpenter, T.P., Moser, J.M. & Romberg, T.A. (1982), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cawley, J.F. (1985). Cognition and the learning disabled. In J.F. Cawley (Ed.), Cognitive strategies and mathematics for the learning disabled (pp. 1-29). Rockville, MD: Aspen Systems Corporation.

Cox, L.S. (1975). Systematic errors in the four vertical algorithms in normal and handicapped populations. Journal for Research in Mathematics Education, 6, 202-220.

Davis, R.B. (1983). Complex mathematical cognition. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Deno, S., & Fuchs, L.S. (1987). Developing curriculum-based measurement systems for data-based special education problem solving. Focus on Exceptional Children, 19(8), 1-16.

Doyle, W. (1988). Work in mathematics classes: The context of students' thinking during instruction. Educational Psychologist, 23(2), 167-180.

Engelhardt, J. M. (1977). Analysis of children's computational errors: A qualitative approach. British Journal of Educational Psychology, 47, 149-54.

Ferrara, R.A. (1987). Learning mathematics in the zone of proximal development: The importance of flexible use of knowledge. Unpublished doctoral dissertation. University of Illinois at Urbana-Champaign.

Ginsburg, H.P., Kossan, N.E., Schwartz, R., & Swanson, D. (1983). Protocol methods in research on mathematical thinking. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Goldman, S.R. (1989). Strategy instruction in mathematics. Learning Disability Quarterly, 12, 43-55.

Harvey, L. F., & Kyte, G. C. (1965). Zero difficulties in multiplication. The Arithmetic Teacher, 12, 45-50.

Hayes-Roth, F., Waterman, D., & Lenat, D. (1983). Building expert systems. Reading, MA: Addison-Wesley.

Hofmeister, A. (1986). Formative evaluation in the development and validation of expert systems in education. Computational Intelligence, 2(2), 65-67.

Hofmeister, A. M., & Lubke, M. M. (1986). Expert systems: Implications for the diagnosis and treatment of learning disabilities. Learning Disability Quarterly, 9(2), 124-132.

Howard, L., & Woodward, J. P. (1990). Misconceptions in subtraction: An analysis of secondary learning disabled students. (Tech. Rep. No. 90-1). Eugene, Oregon: Eugene Research Institute.

Lampert, M. (1986). Knowing, doing, and teaching multiplication. Cognition and Instruction, 3(4), 305-342.

Mandl, H. & Lesgold, A. (1988). Learning issues for intelligent tutoring systems. New York: Springer-Verlag.

McLeod, T., & Armstrong, S. (1982). Learning disabilities in mathematics – Skill deficits and remedial approaches at the intermediate and secondary level. Learning Disability Quarterly, 5, 305-311.

Pellegrino, J.W., & Goldman, S.J. (1987). Information processing and elementary mathematics. Journal of Learning Disabilities, 20, 23-32.

Resnick, L.B. & Omanson, S.F. (1987). Learning to understand arithmetic. In R. Glaser (Ed.), Advances in instructional psychology (volume 3). Hillsdale, NJ: Lawrence Erlbaum Associates.

Resnick, L.B. (1982). Syntax and semantics in learning to subtract. In T.P. Carpenter, J.M. Moser & T. Romberg (Eds.), Addition and subtraction: Developmental perspective. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Resnick, L.B. (1983). A developmental theory of number understanding. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Resnick, L.B. (1988). Treating mathematics as an ill-structured discipline. In R.I. Charles and E.A. Silver (Eds.), The teaching and assessing of mathematical problem solving. Hillsdale, NJ/Reston, VA: Erlbaum and National Council of Teachers of Mathematics.

Resnick, L.B., & Ford, W.W. (1981). The psychology of mathematics for instruction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Roberts, F. (1984). An overview of intelligent CAI systems. Peabody Journal of Education, 62(1), 40-52.

Roberts, G. H. (1968). The failure strategies of third grade arithmetic pupils. The Arithmetic Teacher, 15, 442-46.

Romberg, T.A. (1982). An emerging paradigm for research on addition and subtraction skills. In T.P. Carpenter, J.M. Moser, & T.A. Romberg (Eds.), Addition and subtraction: A cognitive perspective. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schoenfeld, A. (1988). When good teaching leads to bad results: The disasters of "well-taught" mathematics courses. Educational Psychologist, 23(2), 145-166.

Silbert, J., Carnine, D., & Stein, M. (1989). Direct instruction mathematics. Columbus, OH: Charles Merrill.

Sleeman, D. & Brown, J. (1981). Intelligent tutoring systems (pp. 227-282). New York: Academic Press.

Van Lehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. Journal of Mathematical Behavior, 3(2), 3-72.

Van Lehn, K. (1983). On the representation of procedures in repair theory. In H.P. Ginsburg (Ed.), The development of mathematical thinking. Orlando, Florida: Academic Press.

Van Lehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), Learning issues for intelligent tutoring systems. New York: Springer-Verlag.

Wenger, E. (1987). Artificial intelligence and tutoring systems. New York: Morgan Kaufmann.

West, T. A. (1971). Diagnosing pupil errors: Looking for patterns. The Arithmetic Teacher, 18, 467-69.

Woodward, J., Freeman, S., Blake, G., & Howard (1990). TORUS: Computer-based analysis of subtraction. [computer program]. Eugene, Oregon: Eugene Research Institute.

Young, R.M. & O'Shea, T. (1981). Errors in children's subtraction. Cognitive Science, 5, 152-177.

6.0 Subtraction Misconceptions and Remedial Curricula

In order to provide a practical, cost effective method for remedying TORUS diagnostic reports, appropriate lessons in SRA's Corrective Mathematics Program were correlated to common misconceptions. A list of the 10 most common misconceptions appear on the next page. Subsequent pages match the misconceptions to the scope and sequence of the Corrective Mathematics Program, a widely-used remedial curricula for special education students. The lesson sequence has been carefully analyzed so that the learner is provided effective but efficient remediation.

Misconceptions

A. Borrowing - just one column

e.g.

5240	4205	5623
-1630	-1183	-418

B. No borrowing - same number of columns

524	638	841
-130	-104	-110

C. No borrowing vs. adding with carrying

346	528
-212	+136

D. No borrowing - different number of columns

4328	527	5285
-14	-3	-154

E. Borrowing in two columns (the columns are not next to each other)

4264	3022	4020
-1627	-616	-1327

F. No borrowing - the answer to the biggest column(s) is zero which is not written

4263	6825
-4241	-6123

G. Borrowing required in a column just to the right of a column with a zero

6402	8061	400
-1717	-2480	-37

H. Borrowing required in two consecutive columns

5246	8340	4620
-1465	-2490	-395

I. Borrowing required in a column just to the right of two columns that have a 10

3104	6102
-28	-68

J. Borrowing required in a column just to the right of the two columns that have 00

8000	5004
-246	-62

Miscon.	Lesson	TP Task	W B PT
A	11	10	7
B	11	11	8
BC	11	12	9
A	12	10	6
B	12	11	7
BC	12	12	8
A	13	11	8
D	13	12	9
BC	13	13	10
A	14	7	6
A	14	8	7
D	14	9	8
BC	14	11	9
A	15	8	6
A	15	9	7
D	15	11	9
BC	15	12	10

TP = Teachers presentation book

WB = Student workbook

A	16	7	5
A	16	8	6
A	16	9	7
A	16	10	7
BCD	16	11	8
A	17	7	6
A	17	8	7
A	17	9	8
A	17	10	8
BCD	17	11	9
A	18	6	6
BCD	18	7	7
A	19	6	6
BCD	19	7	7
E	20	7	6
A	20	8	7
F	21	6	5
E	21	11	9
A	21	12	10
F	22	7	6
G	22	8	7
E	22	9	8
F	23	8	7

G	23	9	8
E	23	11	9
F	24	7	6
E	24	8	7
G	24	8	8
G	25	6	6
EF	25	10	8
G	26	7	6
E	26	11	8
G	27	9	6
E	27	10	7
G	28	8	6
E	28	9	7
G	29	6	6
E	29	7	7
G	30	8	8
E	30	9	9
G	31	8	8
E	31	9	9
H	32	9	8
EG	32	10	9
H	33	8	7
EG	33	10	9
EGH	34	9	7
EGH	35	9	7

I	41	5	5
EGH	41	8	8
I	42	6	4
EGH	42	8	6
G	43	7	5
J	44	7	-
J	44	8	6
EGHI	44	10	8
J	45	6	6
J	46	4	4
J	48	5	5
EGHI	48	8	8
J	49	6	5
J	50	7	6
J	51	7	6
EGHI	51	9	8
J	52	6	6
J	53	7	6
HIJ	54	8	7
HIJ	55	9	7
HIJ	56	9	7
HIJ	57	10	7
HIJ	58	9	7

7.0 Current Uses of the TORUS Program

TORUS is now being used as a key dependent measure in an OSEP-funded Small Grant for FY 1991-1992. The project uses an information processing framework for understanding the complex and interrelated nature of academic performance of students with learning disabilities. This orientation also enables researchers to more closely assess the nature of individual differences -- and to delineate those with instructional implications. Information processing models have been successfully applied in the areas of reading, writing, and spelling. In mathematics, to date, research has been limited to increasing automaticity in basic arithmetic facts.

TORUS is one of two new assessment techniques and methodologies used in this project. The other, think aloud interview techniques, are also utilized to probe cognitive processes. Our complete analysis of misconceptions is based on Kolligian and Sternberg's (1989) triarchic model of information processing. The model views competent academic performance as influenced by three interrelated components: strength of a knowledge base, strategic processing, and motivation. Each of these components will be investigated using a variety of measures. Finally, the research will be conducted longitudinally.

Findings from this research will have direct implications for the special education teachers' understanding of the mathematics performance of students with learning disabilities and assist them in developing sound instructional programs that directly address students' misconceptions. Finally, what is learned in this research will be applicable to other highly procedural or algorithmic operations in mathematics that students with learning disabilities study (e.g., multiplication, division, fractions, decimals).

8.0 Dissemination

John Woodward has made two major presentations on the TORUS program and its findings. The first was to select education faculty at the University of Michigan in April, 1991. The second was part of a CEC conference symposium on mathematics in April, 1992. In addition, Lisa Howard and Michael Landes presented TORUS findings to the Oregon Conference in February, 1992.

Throughout the project, Apple Computer, Inc., has provided technical assistance and developer discounts on hardware for the project. Neuron Data of Palo Alto, California -- the manufacturer of Nexpert Object™ -- has also provided developer discounts on their product and many hours of free consultation for the project.

A brief article on TORUS was also written for the March 1992 issue of Counterpoint, a national publication for special educators and administrators. A copy of this article appears on the next two pages.

TECHNOLOGY

Computers spot students' math misconceptions

Special educators could one day turn to computerized expert systems to detect the misconceptions that prevent students from grasping math concepts.

If the successes of a prototype for innovative technology is developed commercially, computers could diagnose a student's math errors and identify consistent error patterns, says John P. Woodward, a senior research associate at the Eugene Research Institute (ERI) in Oregon. Specific diagnosis would structure math instruction and benefit student achievement.

In research funded by the Office of Special Education Programs in the U.S. Department of Education, TORUS was launched by researchers at the ERI, with technical support from Apple Computer Inc., and Neuron Data Inc. of Palo Alto, Calif. This project builds on prior work in computer-based systems for analyzing mathematical misconceptions.

Change now

Math instruction is undergoing change nationally because current methods prove deficient for many children, Woodward says. Poor performance is blamed partially on an excessive focus on computation with far too little time devoted to mathematical understanding, problem solving, and reasoning. Some believe math problems begin as early as the first and second grade.

Students often practice computation to an excessive degree, a practice that jeopardizes true mathematical understanding. Some young students view math merely as meaningless symbol pushing. Rather than becoming fluent in the correct method for solving a problem, many students invent their own algorithms, ones that are not tied to conceptual understanding. Thus important concepts such as zero, place value, and so forth are lost or never taught.

The patterns that these misconceptions, or "bugs," take are surprisingly predictable, yet not easily detected by teachers, according to Woodward whose project was created to analyze specific math errors.

TORUS relies on advanced expert system technology to perform diagnoses. Each student answer on a TORUS subtraction assessment is analyzed by its features, such as a borrow involving a zero, or multiple borrows, or asymmetric problems, or one where there are more digits on the top than the bottom.

Comparisons are made to other problems on the test with similar features to uncover a consistent error pattern. TORUS next "predicts" answers to these problems, matching its answer to the students' actual answers. If enough correct matches are made on the same features, then the student is deemed to have a systematic misconception or bug. An computer-generated analysis of 40 subtraction problems requires about two minutes once students complete the test and enter all their answers.

If there are questions about possible misconceptions, the TORUS system generates a second test that is tailored to problems of a particular type or set of features. Results of this test are processed in a similar manner.

The sophisticated nature of this kind of diagnosis is easily underestimated, according to Woodward. Incorrect answers to problems containing a single zero, for example, may look the same to a teacher, but reflect very different underlying misconceptions. Furthermore, the amount of time and concerted effort associated with this kind of diagnosis is impractical for teacher with average classrooms of 25 to 30 students.

So many errors!

Last fall, 80 third graders and 65 fourth graders were given the TORUS test. Some 75 percent of the third graders showed some kind of systematic misconception. For example, 60 percent didn't remember how to borrow. Fourth graders were not much better.

In another study of more than 100 middle school students with learning disabilities that was conducted last year, there were equally dismal performance results. Over half of the students showed weak concepts about borrowing.

Yes, conceptual teaching

TORUS findings are consistent with related research conducted over the last 15 years. For all of the computational practice that students are given in school, the results are discouraging and make the argument for changing math instruction.

It is ironic that while TORUS is designed to analyze computational performance, it underscores the need to teach mathematics conceptually. Ideally, computation problems in subtraction should be thought of as vehicles for teaching concepts and as one way of representing mathematical knowledge.

Ultimately this will prepare students for the work world and the kind of mathematics they will need in an information society.

John Woodward, Eugene Research Institute, 1400 High St., Suite C, Eugene Oregon 97401, (503)342-1553.

Clearinghouse offers help to parents

Vietnam veterans now have a new resource at their disposal when searching for assistive technology to help their disabled children.

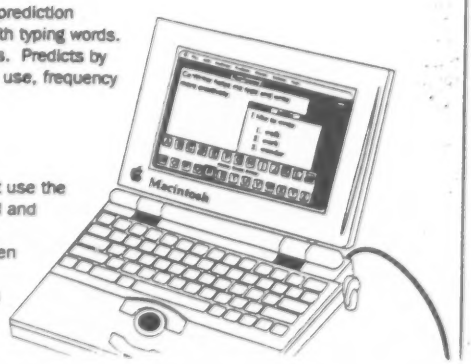
The Access Group, which opened this past fall, is a new information clearinghouse that provides advice and [illegible] U.S. military personnel technology so they can assist others," says Webb. "It's for people who are thinking about, say, an augmented-communication device but need to know what's the best system, how to get service or training or the like."

The project only recently got off the [illegible] after spending its first months

Competitive Tools for Individuals with Physical Disabilities in the Mainstream of Life

Co:Writer™
An intelligent word prediction program for help with typing words. Reduces keystrokes. Predicts by grammar, language use, frequency and recency.

Ke:nx®
For those who can t use the traditional keyboard and mouse. Computer access with onscreen or alternate keyboards or even a single switch.

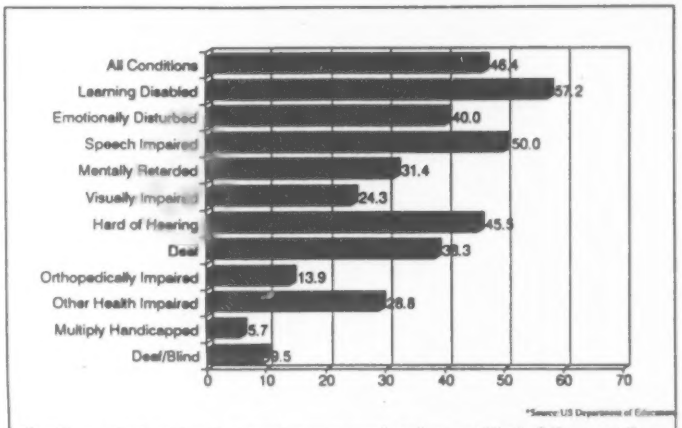


SPRING 1992

Volume 12
Number 3

Counterpoint

Published by the National Association of State Directors of Special Education
as a Service to Teachers and Administrators of Special Education



Youth out of school 1 to 2 years who were employed competitively full or part time.

Study reveals how disabled fare in years after school

By June Behrmann
Senior Editor

In what may be the most important commissioned study released in a decade, the U.S. Department of Education has learned some good, bad, and surprising news about teenage special education students who fail or succeed in their last year of high school and for two years that follow thereafter.

Among factors that affect student performance and outcomes, several reflect back in positive and negative ways to the general and special education administrators and teachers who are responsible for their schooling.

The report provides sufficient information to create a blueprint for change, but as with past data, the data raise hard questions for all educators about the racial, socioeconomic, and other factors that relate to dropping out of school that are now documented in special education.

More than 90 percent of special education secondary students in the study attended regular schools and 86 percent took at least some of their courses in mainstream classrooms. The average amount of time spent in regular education in this study was 56 percent, with some students spending a high of 77 percent, and more seriously impaired students dipping to a low 19 percent. Overall, 17 percent of students took all of their courses in regular education.

On a positive note, nearly 70 percent of the special education students, and particularly deaf students and those with learning disabilities, rose above counterparts in special education and were as productive in the first and second year after high school as nondisabled peers in the general population. While two-year data is promising, once peers earn college degrees, a gap between groups is expected, cautions SRI International's Mary Wagner, who directed the project and notes that only about 1 percent of LD students go after higher education and better outcomes it offers.

In other good news, more and more students

Continued page 17

ED seeks out unneeded, costly federal regs

The time to act might have arrived for educators wanting to break down the regulatory barriers that sometimes interfere with the educating of children with disabilities.

The U.S. Department of Education is requesting public comment on federal regulations that "substantially impede economic growth, are no longer needed, or impose unnecessary costs or burdens."

"This is a real opportunity for educators to convince federal education officials to simplify, eliminate, or redefine some federal regulations," says NASDSE Executive Director Bill Schipper. "The Department of Education is asking for this information rather than we having to convince them to do something."

The Education Department's request is spurred by President Bush's announcement during his State of the Union Address of a 90-day moratorium on new regulations and a requirement that all federal agencies spend the time between Jan. 28 and April 28 reviewing existing regulations to "weed out unnecessary and burdensome government regulations, which impose needless costs on consumers and substantially impede economic growth."

In the Feb. 21 *Federal Register*, the Education Department also asked for comments about regulations in which there are "overlapping, duplicative, inconsistent, or conflicting requirements with other federal agencies, as well as in the department's own regulations." Education Secretary Lamar Alexander also expressed interest in "identifying statutory changes that are needed to reduce impediments to economic growth or to eliminate unnecessary burdensome or costly requirements."

This provides an opportunity to identify the regulatory barriers within Chapter 1 that sometimes interfere with special education/Chapter 1 collaboration, Schipper says. For instance, federal regulations are strict about who has access to equipment purchased with Chapter 1 funds — to the extent that children with disabilities often are barred from using the equipment. Yet the Chapter 1 program is designed to serve many of the same children.

Comments must be received by Education Department officials no later than March 23. "Comments should include the information needed to identify the particular provision at issue, including the program name and a citation to the relevant statute or regulation," the department announced. "Commenters are requested to make their suggestions as specific as possible."

In a related development, department officials still are debating whether the moratorium applies to education regulations and funding priorities. Although the President's order was sent to

Continued page 3

National Association of State Directors
of Special Education, Inc.
1800 Diagonal Road, Suite 320
Alexandria, VA 22314

$1 billion sought for special ed
A coalition of national organizations want Congress to increase federal funding for special education by $1 billion. Page 3.

Professionals join fight against abuse
Special educators are focusing attention on the problem of physical and sexual abuse of children with disabilities. Page 6.

Computers identify kids' weaknesses
Teachers could one day turn to computers to identify the misconceptions that students have about math. Page 15.